IBM z/VSE
5.2

Diagnosis Tools

SC34-2628-03

Note: Before using this information and the product it supports, be sure to read the general information under "Notices" on page 229.

This edition applies to Version 5 Release 2 of IBM® z/Virtual Storage Extended (z/VSE), Program Number 5609-ZV5, and to all subsequent releases and modifications until otherwise indicated in new editions.

This edition replaces SC34–2628–02.

Order publications through your IBM representative or the IBM branch office serving your locality. Publications are not stocked at the addresses given below.

A form for readers' comments is provided at the back of this publication. If the form has been removed, address your comments to:

```
IBM Deutschland Research & Development GmbH
Department 3282
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany
```

You may also send your comments by FAX or via the Internet:

```
Internet: s390id@de.ibm.com
FAX (Germany): 07031-16-3456
FAX (other countries): (+49)+7031-16-3456
```

When you send information to IBM, you grant IBM a non-exclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Contents

Figures

Tables

About This Book

This publication is intended for customers who need to use the diagnosis tools of IBM z/VSE. These tools consist of the various dump types, the Interactive Trace Program, SDAID traces, and the Info/Analysis dump management facility of z/VSE. When to use these tools, and under what circumstances, is explained in z/VSE Guide for Solving Problems.

With interactive tools, you will find examples of the panels and the interactive dialogs that are used to invoke and run the programs. For tools running in batch mode, you will find examples of job control streams. If an explanation of the output is necessary, sample listings are included.

Readers of this publication should be familiar with the operational concept of the IBM z/VSE system.

IBM z/VSE includes diagnosis tools that help you in information gathering and problem diagnosis when a system or program malfunction occurs.

This publication describes the use of these tools.

Who Should Use This Book

This publication addresses primarily the **system administrator**.

Note, however, that any of the following persons may be the first to encounter a problem:

- The system console operator.
- A display station user, including the system administrator.
- An application programmer.
- An application end user.

Most problems, however, will end up with the administrator. Whenever an application program seems to be at fault, the administrator may hand the problem over to the programmer responsible.

How to Use This Book

The conventions for showing the format of job control commands and statements used in the publication z/VSE System Control Statements apply also to this manual.

This publication is divided into the following parts:

- **Dumps of Virtual Storage**

  Which describes the various dump functions in general and shows the file and library environment which is needed to store dumps. The methods to request and to print storage dumps which have been stored on tape or in a dump sublibrary are described. This part contains also the description of the DOSVSDMP utility.
- **Interactive Trace Program**

  The Interactive Trace Program is the tracing tool for z/VSE application programs. This part describes how you can trace the execution of programs running in static or dynamic user partitions.
- **SDAID Trace**

  Contains an overview of the SDAID trace program, describes all trace types and the various methods to initialize them. How you can start, stop, or terminate the initialized traces is also described in this part.
- **Info/Analysis**

  Info/Analysis is the dump viewing and management facility of VSE. This part describes the use of Info/Analysis. It also describes the stand-alone dump analysis routines IJBXCSMG, IJBXDBUG and IJBXSDA.
- **Appendixes**

Contain a description of the symptom record, various display and list aids such as the LVTOC or the LSERV program, and tells how to use some hardware diagnosis aids.

Where to Find More Information

You will need the following IBM publications when diagnosing a problem:

- z/VSE Guide for Solving Problems
- z/VSE Messages and Codes Volume 1
- z/VSE Operation
- z/VSE System Utilities
- z/VSE TCP/IP Support
- *VTAM Diagnosis*

Restriction

If any of these diagnosis tools writes the output to an IBM 3211 printer and this printer's indexing feature is being used, a number of characters may get lost on each line of the output. The system's dump and trace routines, for example, write output records of 120 bytes in length.

To avoid the loss of data, you should load another FCB (forms control buffer) image which disables the indexing feature before requesting the desired printout. For information on FCB loading, refer to the z/VSE System Control Statements.

Summary of Changes

These are the enhancements made available at General Availability of z/VSE 5.2:

- Support of dumps of memory objects generated by the new `OPTION MODUMP` in the interactive interface.

Understanding Syntax Diagrams

This section describes how to read the syntax diagrams in this manual.

To read a syntax diagram follow the path of the line. Read from left to right and top to bottom.

- The ▶▶—— symbol indicates the beginning of a syntax diagram.
- The ——▶ symbol, at the end of a line, indicates that the syntax diagram continues on the next line.
- The ▶—— symbol, at the beginning of a line, indicates that a syntax diagram continues from the previous line.
- The ——▶◀ symbol indicates the end of a syntax diagram.

Syntax items (for example, a keyword or variable) may be:

- Directly on the line (required)
- Above the line (default)
- Below the line (optional)

Uppercase Letters

Uppercase letters denote the shortest possible abbreviation. If an item appears entirely in uppercase letters, it can not be abbreviated.

You can type the item in uppercase letters, lowercase letters, or any combination. For example:

▶▶— KEYWOrd —▶◀

In this example, you can enter KEYWO, KEYWOR, or KEYWORD in any combination of uppercase and lowercase letters.

Symbols

You **must** code these symbols exactly as they appear in the syntax diagram

*****
Asterisk

:
Colon

,
Comma

=
Equal Sign

-
Hyphen

//
Double slash

()
Parenthesis

.
Period

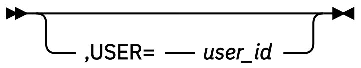
+
Add

For example:

```
* $$ LST
```

Variables

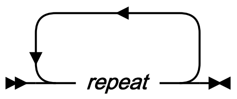
Highlighted lowercase letters denote variable information that you must substitute with specific information. For example:



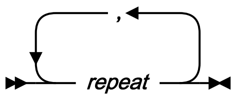
Here you must code USER= as shown and supply an ID for user_id. You may, of course, enter USER in lowercase, but you must not change it otherwise.

Repetition

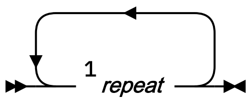
An arrow returning to the left means that the item can be repeated.



A character within the arrow means you must separate repeated items with that character.



A footnote (1) by the arrow references a limit that tells how many times the item can be repeated.

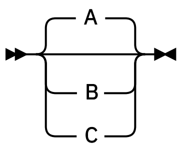


Notes:

[1] Specify *repeat* up to 5 times.

Defaults

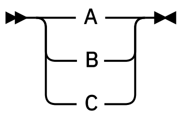
Defaults are above the line. The system uses the default unless you override it. You can override the default by coding an option from the stack below the line. For example:



In this example, A is the default. You can override A by choosing B or C.

Required Choices

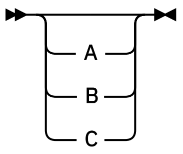
When two or more items are in a stack and one of them is on the line, you **must** specify one item. For example:



Here you must enter either A or B or C.

Optional Choice

When an item is below the line, the item is optional. Only one item **may** be chosen. For example:



Here you may enter either A or B or C, or you may omit the field.

Required Blank Space

A required blank space is indicated as such in the notation. For example:

```
* $$ EOJ
```

This indicates that at least one blank is required before and after the characters $$.

Understanding Physical Addresses and VSE Addresses

From z/VSE 4.3 onwards, z/VSE supports device addresses (that is, *physical addresses*) of up to X'FFFF'. This support is implemented as follows:

- z/VSE applications, messages, commands, and so on, do not address a device by the physical address (**pcuu**), but instead by the *VSE address* (**cuu**).
- VSE addresses are in the range from X'000' to X'FFF'.
- To each physical address (pcuu) there is a corresponding VSE address (cuu).
- If the physical address is less than or equal to X'FFF', the VSE address (cuu) is equal to the physical address (pcuu).
- If the physical address is higher than X'FFF' (and therefore outside the range of VSE addresses), the physical address (pcuu) and VSE address (cuu) will be different.

CP commands (under z/VM) always use *physical addresses*.

z/VSE jobs, commands, dialogs, and messages use *VSE addresses*. However, in specified cases (for example, when using the QUERY IO command) *physical addresses* might be used.

Note: Throughout the z/VSE documentation, the term *address* of a device (used on its own) always refers to the *VSE address*.

To obtain the VSE address of a device that corresponds to a physical address, you can use the QUERY IO command. For example, to display the VSE address (cuu) of a tape drive that has the physical address (pcuu) 3A61, you would enter at the z/VSE console:

```
query io,cuu=3A61
AR 0015  VSE ADDR   PHYSICAL ADDR   DEVICE CLASS
AR 0015       A61            3A61   TAPE
AR 0015 1I40I  READY
```

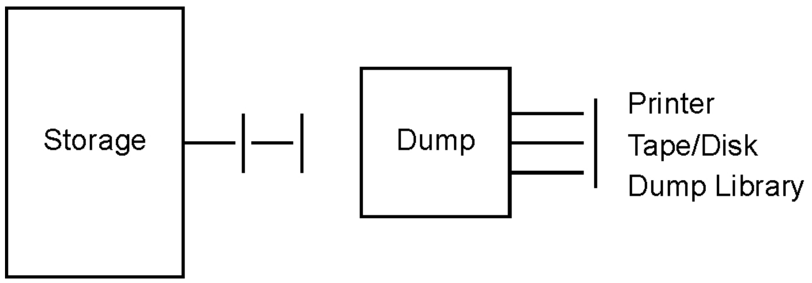
The VSE address shown above is A61.

You can similarly use the QUERY IO command to obtain the physical address of a device that corresponds to a VSE address.

For further examples of using physical addresses of up to X'FFFF', refer to the description of the ADD command in the z/VSE System Control Statements.

Part 1. Dumps of Virtual Storage

You may face system conditions in which you want to know what the contents of your system's storage is. For this, the storage data can be read out, saved in a library or on a tape, or can be printed. This processed storage data is called a **dump**.



This part of the publication describes how to retrieve a dump and how to use the saved dump for problem determination.

Which of the shown methods you use to retrieve information for problem determination depends on the error situation. For example, in case of a system wait or a system loop the Standalone Dump Program would be the appropriate tool to save or print the storage contents.

Chapter 1. General Description

This topic describes the various types of dump, the functions you use to create dumps in general, and how to define dump sublibraries.

The types of dump described in this topic are:

- The **ABEND dump**, initiated by
  - the system ABEND handling routines,
  - the programmer, issuing the macro DUMP,
  - the operator entering the CANCEL command;
- The **DUMP command dump**, initiated by the operator;
- The **Stand-alone dump**, initiated by the operator;
- The **SDAID dump**, initiated by the operator.

Dump Contents Overview

The output of the DUMP command, ABEND dump and Standalone Dump Program contains two major parts.

- The symptom records.
- The data records.

The amount of information which is stored in these dump records depends on the function which requests the dump. Note that pages containing only zeros are not dumped explicitly.

Figure 1 on page 3 gives an overview of a dump, which can reside either in a dump sublibrary or in a dump file on tape or disk.

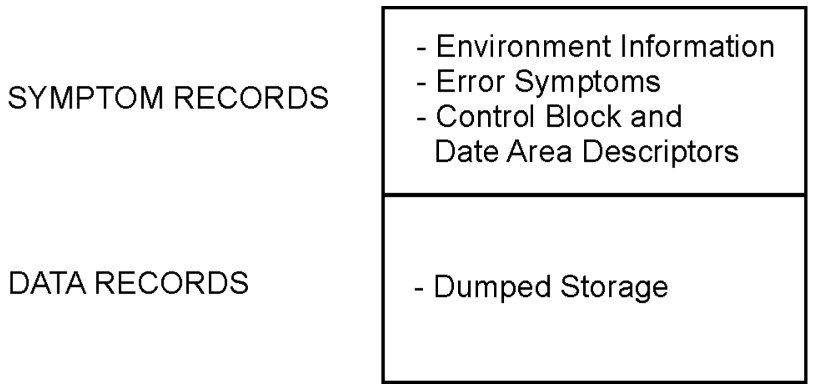

SYMPTOM RECORDS
- Environment Information
- Error Symptoms
- Control Block and Date Area Descriptors
DATA RECORDS
- Dumped Storage

Figure 1. Overview: Dump Contents

The symptom records are built by the component which produces the dump. They contain information to format the dump data later on. The symptom records are described in Appendix A, "Symptom Records Overview," on page 213.

The ABEND Dump Function

The ABEND dump function is internally called when the system detects an ABEND condition or when a CANCEL command has been given.

Overview of ABEND Dump Function

What is an ABEND

ABEND stands for ABnormal END of task. This means that a program (task) is terminated prior to its completion because of an error that could not be resolved by system recovery facilities.

What is an ABEND Dump

The system's ABEND dump function is called by VSE/Advanced Functions:

- When an ABEND (abnormal termination) occurs;
- When a CANCEL command is issued.

When the function is called, it provides a dump of the storage areas in which the program was running.

Figure 2 on page 4 shows that:

- The ABEND dump function is activated when an ABEND condition occurs;
- The output from the function is controlled by job control options. These are specified in STDOPT or OPTION statements.
- The options determine:
  - The contents of the dump;
  - To which I/O device the dump is written.

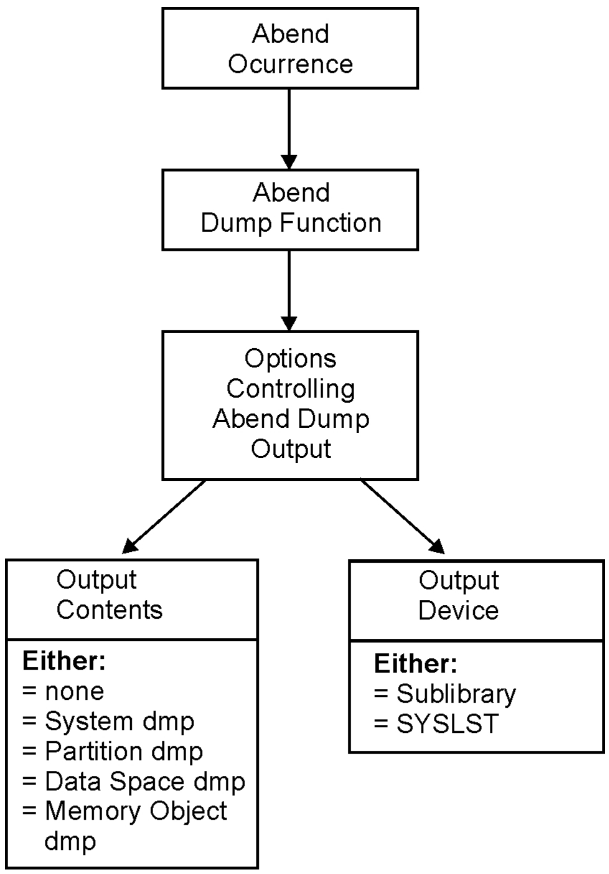


Figure 2. Overview: The ABEND Dump Function

The following ABEND dumps are shown in Figure 2 on page 4 under ‘OUTPUT CONTENTS’:

- The **System Dump** dumps the whole supervisor area and the dump symptoms besides the partition area.
- The **Partition Dump** includes only selected VSE/Advanced Functions control blocks and the dump symptoms in addition to the partition area.

- The **Data Space Dump** includes a dump of one or more data spaces.
- The **Memory Object Dump** includes a dump of one or more memory objects.

The output of the dump is either written into a dump sublibrary or on a printer device assigned to SYSLST.

Note: Memory objects are *only* dumped to SYSLST.

Activation of the ABEND Dump Function

The ABEND dump function is activated when

- A program or task running in one of the system's partitions comes to an **ABnormal END**, and **no AB exit**routine is active. The macro DUMP activates the ABEND handling routines, too. See "Options to Control the ABEND Dump" on page 21.
- A **CANCEL command** is issued by the operator for one of the operating system's partitions. See "Requesting a Dump by the CANCEL Command" on page 23.

If the activation of the ABEND dump function leads to a dump writing operation (depending on the active job control options), the storage contents are dumped

- Before any end-of-job routine is executed.
- Before any of the attached subtasks is terminated.

Contents of the ABEND Dump Output

The output of the ABEND dump function (either in the sublibrary or on SYSLST) contains:

- A dump symptom part, which is always included.
- A system dump or a partition dump, depending on the options active at the time the dump was taken.
- A data space dump, if the corresponding option was specified.
- A memory object dump, if the corresponding option was specified.

Symptom Part of the ABEND Dump

This part of the output contains

- Control data from the symptom records, like information about the environment or the failure. For a description of the symptom records, see Appendix A, "Symptom Records Overview," on page 213.

System Dump

The system dump, which is produced if OPTION DUMP or STDOPT DUMP=YES is active, contains the following information besides the symptom part:

- The ending task PSW, general purpose registers, access registers and floating point registers.
- The entire supervisor area.
- The areas containing z/VSE control blocks listed in Figure 3 on page 6.
- The allocated portion(s) of the system GETVIS area.
- If the error occurred in the SVA, that part of the SVA which holds the phase responsible for the ABEND.
- The partition for which the ABEND dump function is active including areas acquired dynamically within the partition by GETVIS macros in your program.
- The dynamic space GETVIS area for dynamic partitions.

Partition Dump

A partition dump is produced when option OPTION PARTDUMP or STDOPT DUMP=PART is active. The dump output includes the following system areas besides the symptom part:

- The ending task PSW, general purpose registers, access registers, and floating point registers;

- The LOWCORE (low address storage);
- The areas containing VSE control blocks listed in Figure 3 on page 6;
- The partition for which the ABEND dump function is active including areas acquired dynamically within the partition by GETVIS macros in the program;
- If the error occurred in the SVA, that part of the SVA which holds the phase responsible for the ABEND;
- The logical transient area (LTA), if the error causing the dump to be taken occurred in a task owning the LTA.

Data Space Dump

If OPTION DSPDUMP or STDOPT DSPDUMP=YES is active, a data space dump is to be taken in case of an abnormal program end. If the ABEND routine finds out that the failing program has access to a data space, it takes a dump of that space and enters it as a separate library member in the same dump library. The failing program must be in access register mode, and at least one of the access registers must contain the ALET (access list entry token) of that data space. The number of different ALET pointers in the access registers determines how many data spaces will be dumped.

The ABEND routine dumps an area of at least 4K of storage on either side of the address(es) pointed to by the matching general register(s). However, if the size of the data space does not exceed 128K of storage, the whole data space is dumped.

```
SUP       Supervisor
BG        Background partition
GETVIS24  24-bit GETVIS area
GETVIS31  31-bit GETVIS area
COMREG    Partition's communication region
SYSCOM    System communication region
PUBTABLE  Physical unit block table
PUBOWN    PUB ownership table
PUB2TAB   Physical unit block extension table
LUBTAB    Partition's logical unit block table
LUBEXT    Partition's LUB table extension
DIBTAB    Partition's disk information block
PIBTAB    Partition information block
PIB2TAB   Partition information block extension
PCB       Partition control block
AF-TIB    Task information block
AF-TCB    Task control block
LOADLIST  Partition's phase load trace table
LPT       Library pointer table
LDT       Library definition table
SDT       Sublibrary definition table
EDT       Extent definition table
DDT       Device definition table
LIB_ANC   Library anchor table
L-TASK-R  Librarian task LOT-row
LOTPOOL   Library offset table pool
```

Figure 3. VSE Control Blocks in System Dump

Memory Object Dump

When an "abnormal program end" (ABEND) occurs, if `OPTION MODUMP` or `STDOPT MODUMP=YES` is active then z/VSE will create a *memory object dump* of 4 KB of both sides of the failing address.

A *memory object dump* will be created when *all* of the following conditions apply:

- The failing program is running in 64-bit mode when the abnormal program end occurs.
- The current primary address space owns *private memory objects* (defined via an IARV64 GETSTOR request) or *shared memory objects* (defined via an IARV64 GETSHARED request). For details about IARV64 requests, refer to z/VSE System Macros Reference.
- At lease one general register contains a 64-bit address within the range of a memory object.

For details about the `OPTION MODUMP` and `STDOPT MODUMP=YES` options, refer to the z/VSE System Control Statements.

For each matching general register, an area of 4 KB of storage on either side of the 64-bit address contained in the register is dumped. If the 64-bit address is located near a boundary of a memory object (which results in less than 4 KB of storage on one side), the dump is only taken to the *boundary* of the memory object.

An example of a memory object dump that has been produced by the ABEND routine is shown below.

```
MEMORY OBJECT DUMP SYMPTOM RECORDS:
 ADDRESS_SPACE=BG
 REG=02 0000000180302000
 START_ADDRESS=0000000180300000
 END_ADDRESS=00000001804FFFFF
 SHARED=NO
 FETCH_PROTECTED=YES
 STORAGE_KEY=1

DUMP BEGIN: 0000000180301000
DUMP END:   0000000180302FFF

DUMP OF MEMORY OBJECT
0000000180301000 C7C5E3E2 E3D6F5C7 00000000 00000002 C2C70000 00000000 00000000 00000000   10   GETSTO5G........BG..............
0000000180301020 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000        ................................
0000000180301040 TO 0000000180301FFF  SUPPRESSED LINE(S) SAME AS ABOVE ...
0000000180302000 PAGE(S) NOT USED
NO SHARED MEMORY OBJECTS ARE AVAILABLE FOR DUMP
```

Figure 4. Example of a Memory Object Dump

For further details about the symptom records of a memory object dump, refer to the information provided under "REQUIRED SYMPTOMS FOR A MEMORY OBJECT DUMP" in "Symptom Part Description" on page 183.

For details of how memory objects can be included or excluded in a stand-alone dump, see "Partition, Data Space, and Memory Object Dump Files" on page 9.

The DUMP Command

You can request a dump of parts of the virtual storage with the attention routine command DUMP.

For a detailed description of the DUMP command, refer to the publication z/VSE System Control Statements.

The Stand-Alone Dump (SADUMP) Program

If your system entered a hard or soft wait state or is in a continuous loop, no normal system operation is possible. In this case you can invoke the Standalone Dump Program to get information about the problem. The Standalone Dump Program records the supervisor and the SVA in one file and the page manager address spaces and the selected partitions and data spaces in separate files on one or more stand-alone dump tapes or on a disk device. (On disk, these files reside in one physical extent containing several 'logical' files.)

The output device on which the Standalone Dump Program is created must be a tape or disk unit. The Standalone Dump Program writes its dump output on one or more tapes or on a disk device. It is not possible to write the dump output directly on a line printer.

The Standalone Dump Program dumps selected parts of virtual storage of your VSE system on tape or disk. The // OPTION SADUMP job control option allows to include important pieces of virtual storage into the stand-alone dump. It is usually not necessary or practical to dump the complete system; DUMP command dumps and partition dumps should be used when possible.

Before you can use the Standalone Dump Program, it has to be created with the DOSVSDMP utility. See "Creating the Standalone Dump Program" on page 27.

The creation of the Standalone Dump Program should be done shortly after system installation via IUI panels in order to have the program available in case of a system error.

After the dump is taken, the operator has to perform a manual IPL from SYSRES. If the dump is on SYSRES, no manual IPL is required.

For a description of how to request a stand-alone dump, see "Taking a Stand-Alone Dump" on page 23.

Support of Integrated Console by SADUMP Program

The Standalone Dump Program (SADUMP) supports the integrated console as system console in addition to 3215 and 3270 type devices. The selection of the system console depends on device availability and the IPL load parameter.

The selection criteria are:

1. If a console is specified as load parameter, the system will route messages to that preferred device.
   - In case the integrated console is specified in the load parameter, SADUMP will route messages to the integrated console.
   - In case a local console is requested in the load parameter, or the integrated console is not available, SADUMP will route messages to the local console.
2. If the Standalone Dump Program is activated without specification of a communication device type, SADUMP will route messages to the device which is found in the SYSCOM.
   - If this device is not operational, SADUMP will route messages to the integrated console.
   - If an integrated console is not available, SADUMP will abnormally terminate.

IPL Load Parameter

The IPL load parameter must be used to specify the preferred communication device.

To determine the communication path, SADUMP first analyzes the load parameter. If the hardware does not support the load parameter, selection of the communication device is determined during the creation of the Dump program by the DOSVSDMP utility.

If the load parameter is specified, its first byte indicates the **console type**.

For details of the IPL load parameter, refer to the publication z/VSE System Control Statements.

Output of the Standalone Dump Program

The Standalone Dump Program stores the dump information on the tape from where it has been loaded or on a disk extent. It produces a main dump file of the system areas and additional dump files for the page manager address spaces and for each partition and data space to be dumped.

Main Dump File

The 'main' dump file is always file 3 on tape (files 1 and 2 are used by the system) or file 1 on disk. The Standalone Dump Program writes the following information into the main dump file:

- The symptom record, which holds information on the hardware and software environment, error symptoms, and control block locators.
- The dump data, which consists of retrieved pages from processor storage, or from the page data set. It includes the shared area (supervisor, system GETVIS area, and SVA) and control block locators for supervisor control blocks.
- If certain system information needed for accessing the page data set is not available, you get a dump of the data in processor storage only.
- The last 200 messages from the hardcopy file.

The main dump file can be onloaded into a VSE dump sublibrary from which it can be processed by Info/Analysis. The Info/Analysis exit routines IJBXCSMG, IJBXDBUG and IJBXSDA can be invoked to analyze the main dump file.

Page Manager Address Space (PMRAS) Dump Files

The first file (PMRAS-R) contains real storage areas which are used by the Page Manager but are not mapped in any of the virtual spaces. The following files (PMRAS-nn) contain the Page Manager Address Spaces (segment tables, page tables etc.), where nn is the space id.

Partition, Data Space, and Memory Object Dump Files

The job control OPTION SADUMP=n|([n],m)|([n],[m],o) statement specifies the priority in which the partition (n), any owned data spaces (m), or private memory objects (o) should be dumped in a stand-alone dump. The priority setting can be 0 to 9, with 9 being the highest priority and 0 indicating that no dump is taken. The default is ’0’.

The job control command STDOPT SADUMP=n|([n],m)|([n],[m],o) specifies the priority in which **all** partitions/data spaces/private memory objects in the system should be dumped in a stand-alone dump, unless overridden for a specific partition by a corresponding OPTION SADUMP statement.

Examples:

```
F1 ... SADUMP=(5,5)
F3 ... SADUMP=(0,0,5)
F2 ... SADUMP=4
```

Dumps by priority: F1 partition, F1-owned data space(s), F3-owned memory objects, F2 partition.

```
F1 ... SADUMP=(5,3)
F2 ... SADUMP=4
F3 ... SADUMP=(,9)
```

Dumps by priority: F3-owned data space(s), F1 partition, F2 partition, F1-owned data space(s).

Note that for stand-alone dumps to disk, the Standalone Dump Program stops dumping when the dump data set becomes full. Therefore, it is possible that one or more of the partitions, data spaces, or memory objects with SADUMP not equal to 0 will not be dumped, or that the last dump file may be incomplete. This does not apply for stand-alone dumps to tape, since the output can be written to several tapes.

The job control command STDOPT SADMPSMO specifies whether or not the shared memory object dump file `SHARED-MEMORY_OBJ` should be included in a stand-alone dump. The default is ’No’.

For details of how to use the OPTION SADUMP, STDOPT SADUMP, and STDOPT SADMPSMO statements or commands, refer to "OPTION (Set Temporary JC Options)" and "STDOPT (Standard JC Options)" in the publication z/VSE System Control Statements.

All dump files can be processed via DOSVSDMP which allows the contents of an appended dump file to be printed on SYSLST.

All dump files can also be onloaded into a VSE dump sublibrary. From the VSE dump sublibrary, the partition and data space dumps can be processed by the Info/Analysis program. You can display the symptom string or print selected parts of the storage dump.

The Info/Analysis exit routines IJBXCSMG, IJBXDBUG and IJBXSDA cannot be invoked to analyze the appended dump files.

A description of how to print the Standalone Dump Program output can be found under “Printing a Dump Stored on Tape or Disk” on page 202.

The SDAID Dump

The SDAID program can also be used to dump virtual storage. You may use this program for example if you need a dump of a certain part of storage at a defined event.

For a short description of this SDAID function, see “Requesting a Dump on Event (SDAID Dump)” on page 24.

Dump Requested by Macros

A dump of virtual storage can also be requested through dump macros.

For a short description of this method of requesting a dump, see “Requesting a Dump from a Program” on page 25.

Info/Analysis

Info/Analysis is a component of VSE. It is a tool to:

- Manage the dump files
- Print or display dump information.

With Info/Analysis, you can simplify the task of using dump data to solve software problems. Info/Analysis assists you in this task through the following functions:

- Dump management - to list the dumps being managed by Info/Analysis, to add or delete dumps from that list, and to delete dumps from the system.
- Dump symptoms - to display problem failure information collected by the dumping component and by subsequent analysis routines.
- Dump viewing - to display dump data in hexadecimal and character format, to format control blocks and other dump data that may be relevant to the problem, to invoke dump analysis routines, and to display the results of those routines.
- Dump offload - to copy a dump to tape for later retrieval.
- Dump onload - to copy a dump to a dump sublibrary (a stand-alone dump for example).

You enter input either from SYSIN or from SYSLOG. Output always goes to SYSLST. For an example of a job to invoke Info/Analysis, see Figure 69 on page 176.

Chapter 2. Maintaining the Dump Library and File Environment

Various files are used to process and evaluate dumps stored either on a tape or disk volume or in a dump sublibrary.

The Library and Files Required to Process Dumps

The libraries and files required to process and use dump information are:

1. The dump sublibraries (in the library SYSDUMP)
2. The dump management file (for Info/Analysis)
3. The external routines file (for Info/Analysis).

The SYSDUMP Sublibraries

The system uses the defined dump sublibraries to store dumps for later processing. These sublibraries are also used to onload dumps which have been stored on tape either by the system's dump functions or by a previous Info/Analysis offload operation.

Dumps can be processed using the print, analyze, and management functions of the Info/Analysis program once the dumps have been onloaded into a dump sublibrary. How the SYSDUMP library is defined and used is described in the following section.

Purpose of the SYSDUMP Library

The library named SYSDUMP is used to store the various dump types for further processing. It contains one or more dump sublibraries. Each dump sublibrary should be assigned to one partition and may contain one or more dumps. A separate dump sublibrary is used for all dynamic partitions. Figure 5 on page 12 gives an overview of the SYSDUMP library concept.

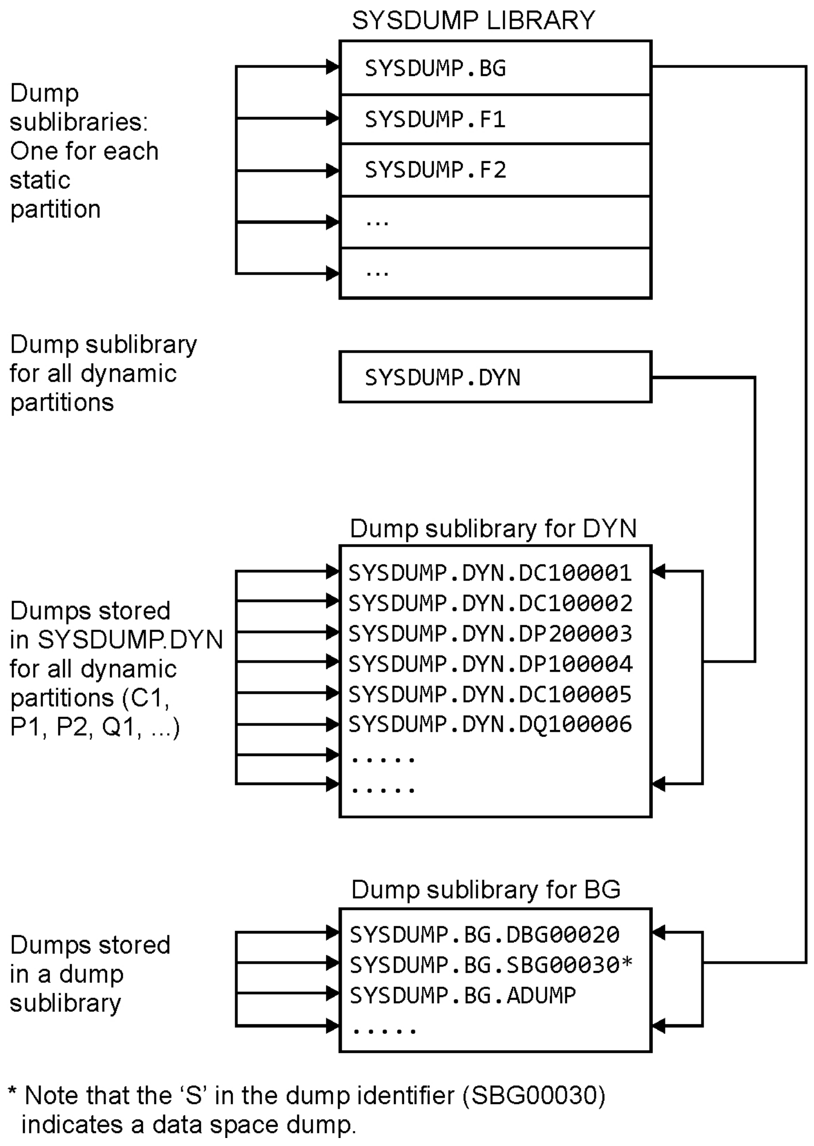


Figure 5. The SYSDUMP Library Concept

These dump sublibraries are used by the system and by you.

VSE/Advanced Functions stores dumps for later processing from

- ABEND events
- CANCEL commands.

You can use the dump sublibraries to onload dumps which have been stored on tape or disk in order to process them with Info/Analysis functions. You may use the dump sublibraries to store the following:

- DUMP command dumps
- Stand-alone dumps (from tape or disk)
- Dumps which have been offloaded to tape.

Establishing the Dump Sublibraries

Before dumps can be stored, the dump sublibraries have to be created. The following describes, what job control label information is required for the SYSDUMP library and how the dump sublibraries can be defined via the librarian program LIBR.

The following *requirements* have to be met if you want to use the dump library and its sublibraries:

1. If the dump library is located in BAM space, you have to specify DLBL and EXTENT labels for the library SYSDUMP.
2. If the dump library is located in VSAM space, you have to specify the DLBL label for the library SYSDUMP.
3. The library SYSDUMP and its sublibraries have to be defined with the LIBR program.
4. LIBDEF statements have to be given.
5. The SYSDUMP option has to be set in order to get ABEND dumps written into the dump sublibraries.

Label Information for SYSDUMP

Figure 6 on page 13 shows an example of the label and extent information you have to submit if you want to define the dump library SYSDUMP.

The standard label area should be used to store this information in the SYSDUMP Library in BAM space or in VSAM space:

```
 ⋮
// DLBL SYSDUMP,'VSE.DUMP.LIBRARY',1999/365,,DSF
// EXTENT SYS010,,1,0,3150,600
 ⋮
```

Figure 6. Example: Labels for the SYSDUMP Library Stored in BAM Space

```
 ⋮
// DLBL SYSDUMP,'VSE.DUMP.LIBRARY',,VSAM,                                  X
               CAT=IJSYSCT,                                               X
               DISP=(OLD,KEEP)
 ⋮
```

Figure 7. Example: Labels for the SYSDUMP Library Stored in VSAM Space

Notes:

- IBM recommends securing the dump library. Securing the dump library prevents overwriting of part of the file(s) as a result of a faulty response to an OVERLAPPING EXTENT message. For information about using the access control function, refer to the topic "Protecting Data" in the publication z/VSE Guide to System Functions.
- From z/VSE 5.1 onwards, an initial installation of z/VSE will create the SYSDUMP library in VSAM space.

Defining the Dump Library

You define the dump library (normally named SYSDUMP) with the LIBR program. Figure 8 on page 13 shows an example of such a definition.

```
// JOB DEFINE
// EXEC LIBR
DEFINE L=SYSDUMP
DEFINE S=SYSDUMP.BG -
         SYSDUMP.F1 -
         SYSDUMP.F2 -
         SYSDUMP.F3 -
         SYSDUMP.F4 -
           ...
         SYSDUMP.FB -
         SYSDUMP.DYN REUSE=IMM
...
/*
/&
```

Figure 8. Example: Defining SYSDUMP with the LIBR Librarian Program

LIBDEF Statement for Dump Sublibraries

To get the dumps stored into the sublibrary assigned to the partition, the ASI Job Control procedure for each partition should contain a LIBDEF statement as shown in Figure 9 on page 14. In the example given in Figure 9 on page 14 a dump sublibrary is connected to the BG partition.

```
// LIBDEF DUMP,CATALOG=SYSDUMP.BG,PERM
```

Figure 9. Example: LIBDEF Statement for a Dump Sublibrary

Chapter 3. Handling Dumps

Options to Activate Dump Writing

The system writes the output of an automatically invoked ABEND dump into the dump sublibrary for the partition if you submit either of the following statements:

```
// STDOPT SYSDUMP=YES
// OPTION SYSDUMP
```

With the // STDOPT SYSDUMP statement you request the system to write dumps of the next and all subsequent jobs or job-steps into the dump sublibrary for the particular partition until the SYSDUMP option is deactivated. The STDOPT statement must be given in the BG partition and is active for all partitions.

You can display the current settings for the *permanent options* using the QUERY STDOPT command.The // OPTION statement is active only for the duration of the particular job (this is the *temporary option).* After EOJ, the permanent option given in a previous STDOPT statement will be active again.

Options to Deactivate Dump Writing

The SYSDUMP option is deactivated by:

```
// STDOPT SYSDUMP=NO
// OPTION NOSYSDUMP
UNBATCH (to deactivate the partition)
LIBDROP DUMP,PERM
```

Identifying the Stored Dumps

Once the dump library and dump sublibraries have been defined, dumps from various sources can be stored there. The dumps stored by the ABEND dump routines have an identifier of the following format:

```
SYSDUMP.partition_id.nnnnnnnn
```

SYSDUMP
Dump library name.

partition_id
Sublibrary name, normally the partition identifier, like BG or F3 or, for dynamic partitions, DYN.

nnnnnnnn
Dump identifier of the form:

Dppnnnnn
for address space dumps, or

Sppnnnnn
for data space dumps.

pp = partition identifier of the static or dynamic partition.

n = integers between 0 and 9 which are maintained by the system automatically with every new store dump operation.

For example:

```
SYSDUMP.F4.DF400002
```

is the name of a dump residing in the dump sublibrary for the F4 partition of the library SYSDUMP, with the identifier DF400002.

Note: When you onload a dump into the dump library via Info/Analysis, you select a dump name by your choice. The rules for creating a dump name are explained in "Recommendations (Restrictions) for the Generation of Dump Names" on page 178

Sending Dumps to IBM Support Electronically

Dumps are normally stored by z/VSE in the VSE dump library. However, you may need to transmit dumps *electronically* to other locations, such as to IBM Support (previously, you were required to send dumps to IBM on a physical tape).

1. **Locate the Dump Location on z/VSE**. If the dump is a stand-alone dump, it will be stored on tape. If the dump is a system dump (for example, an Attention Routine dump), it will be stored on a disk or tape. In both cases, you must upload the dumps to the z/VSE dump library.
2. **Upload the Dumps to the z/VSE Dump Library**. You use the *Storage Dump Management* dialog (Fastpath 43) to do so. For details, refer to the publication z/VSE Guide for Solving Problems. For a stand-alone dump, there will be multiple dumps consisting of one for each selected partition, dataspace, or memory object, and one for the supervisor and SVA.

   If you need to upload dumps of *memory objects*, the INFOANA utility and Dump Management dialog *cannot* be used. Instead, use skeleton SKDMPONL in VSE/ICCF Library 59 to onload a dump from tape (for details, see "Uploading Large Dumps From a Standalone Dump Tape" on page 18). For a stand-alone dump on disk, transfer the contents of the disk *directly* to IBM.
3. **Format and Print Dumps (Optional)**. On the z/VSE dump library, you might be required to use the *Interactive Interface* tools to format these dumps (for example CICS® dumps). However, in most cases IBM Support will require *unformatted* dumps.
4. **Download the Dumps From z/VSE to Your PC**. To download dumps from z/VSE to your Personal Computer (PC), you use the *File Transfer* utility of the Interactive User Interface (IUI). For the example of "Identifying the Stored Dumps" on page 15, you would enter this command at the PC:

   ```
   receive DFH400002.dump a:DFH400002 dump (file=lib l=sysdump s=f4 binary
   ```

   In this example, "a:" is the emulation session where you are signed on to CICS.

   As an alternative to using the File Transfer utility of the IUI, you can use the FTP of TCP/IP to transfer dump to transfer the dump from the z/VSE dump library to your PC. For the example used in this procedure, a transfer using FTP would appear like this:

   a. **Define the z/VSE Dump Library to TCP/IP**:

      ```
      DEFINE FILE,TYPE=LIBRARY,DLBL=SYSDUMP,PUBLIC='SYSDUMP',ALLOWSITE=NO
      ```

   b. **Start the File Transfer (in Binary)**:

      ```
      C:\>ftp 9.164.155.2    <----- Your IP address
      Connected to 9.164.155.2.
        ⋮
      User (9.164.155.2:(none)): sysa   <----- your user id
      331 User name okay, need password.
      Password:              <----- your password
      200 Command okay.
      ftp> cd sysdump
      250 Requested file action okay, completed.
      ftp> cd f4             <----- Sublib where your dump resides
      250 Requested file action okay, completed.
      ftp> bin               <----- switch to binary mode
      200 Command okay.
      ftp> get DF400002.dump <----- Name of the dump
      200 Command okay.
      150-About to open data connection
          File: SYSDUMP.F4.DF400002.DUMP
          Type: Binary Recfm: S  Lrecl:  4096
          CC=ON  UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON
      150 File status okay; about to open data connection
      ```

```
226-Bytes sent: 23,273,920
    Records sent:           711
    Transfer Seconds:         6.91 (  3,788K/Sec)
    File I/O Seconds:         5.30 (  4,545K/Sec)
226 Closing data connection.
ftp: 23273920 bytes received in 7,51Seconds 3098,64Kbytes/sec.
```

5. **Send the Dump to IBM Support**. Depending on the size of the dump stored on your PC, you might need to compress the dump using the PKZIP utility. To send the dump to IBM Support, you should send the dump to an IBM FTP server using the FTP of TCP/IP. For the example used in this procedure, you would send the dump to the public server at IBM Boulder using these commands:

```
C:\>ftp testcase.boulder.ibm.com   <----- IBM Boulder IP address
Connected to testcase-blue.boulder.ibm.com.
  :
User (testcase-blue.boulder.ibm.com:(none)): anonymous
331 Guest login okay, need password.
Password:                 <----- your Internet e-mail address
200 Command okay.
ftp> cd /vse/toibm        <----- cd vse [enter] cd toibm [enter]
250 Requested file action okay, completed.
ftp> bin                  <----- switch to binary mode
200 Command okay.
ftp> put DF400002.dump    <----- Name of the dump
200 Command okay.
150-About to open data connection
    File: SYSDUMP.F4.DF400002.DUMP
    Type: Binary Recfm: S  Lrecl:  4096
    CC=ON  UNIX=OFF RECLF=OFF TRCC=OFF CRLF=ON
150 File status okay; about to open data connection
226-Bytes sent: 23,273,920
    Records sent:           711
    Transfer Seconds:         6.91 (  3,788K/Sec)
    File I/O Seconds:         5.30 (  4,545K/Sec)
226 Closing data connection.
ftp: 23273920 bytes received in 7,51Seconds 3098,64Kbytes/sec.
```

Mailing Dumps That Are Stored on Tape to IBM Support

Dumps are normally stored by z/VSE in the VSE dump library. However, you may need to mail dumps that are stored *on tape* to other locations, such as to IBM Support.

1. **Locate the Dump Location on z/VSE**.
   - If the dump is stored on disk, go to Step 2 below.
   - If the dump is stored on tape and you wish to format the dump, go to Step 2 below.
   - If the dump is stored on tape and you do **not** wish to format the dump, mail the tape to the address provided by IBM Support.
2. **Upload the Dump From Disk to the z/VSE Dump Library**. You use the *Storage Dump Management* dialog (Fastpath 43) to do so. For details, refer to the publication z/VSE Guide for Solving Problems.

   If you need to upload dumps of *memory objects*, the INFOANA utility and Dump Management dialog *cannot* be used. Instead, transfer the contents of the disk *directly* to IBM.
3. **Format and Print the Dump (Optional)**. On the z/VSE dump library, you might be required to use the *Interactive Interface* dialog to format these dumps (for example CICS dumps). However, in most cases IBM Support will require *unformatted* dumps.
4. **Download the Dump From the z/VSE Dump Library to Tape**. To download a dump from the z/VSE dump library to tape, you use the *Interactive Interface* dialog.
5. **Mail the Tape to IBM Support**. You should mail the tape to the address provided by IBM Support.

Handling a Dump Library Full Condition

Information is written to the dump sublibraries when:

1. A dump is taken automatically by the system;

2. A dump is stored in a sublibrary via the onload process of Info/Analysis dump management;
3. Dumps are examined with an Info/Analysis analysis routine function.

In all three cases, space is needed in the dump sublibrary for page maps and analysis information, in addition to the dumps themselves. How the system reacts to a library-full condition depends on which routine caused the condition, and what kind of information was being written at the time.

If the library becomes full while:

- The system is writing an ABEND dump into it, the whole dump is printed on SYSLST and a dump-library-full information message is issued on SYSLOG;
  - If you want to *ignore* the dump (rather than printing it on SYSLST) you must specify *SYSDUMPC* instead of SYSDUMP. If the dump does not fit into the dump library, it will be printed.
  - Refer to the IBM publication z/VSE System Control Statements for details of how to specify the SYSDUMPC option.
- The Info/Analysis dump management function is writing a dump, the dump is flagged “to be onloaded”. In spite of this flag, the dump may have been stored in the sublibrary. This can happen when the library-full condition arises while additional information is being stored after the dump itself has been written.
- An Info/Analysis dump viewing function is being used, the function fails.

In all three cases, the amount of free space in the sublibrary is kept as it was before the dump write operation was started.

- You can clear sublibrary space to make room for new dumps by deleting dumps that are no longer required. How to delete a dump is described under “DELETE - Delete Current Dump” on page 180.

  Note: Do not delete a dump with a delete function other than the Info/Analysis delete function.
- You can also send dumps to the *dump archive* (sub-library PRD2.DUMP). This archive resides in VSAM space, and is therefore *automatically* extended by VSAM. To set your z/VSE system to use the dump archive, use the dialog *Storage Dump Management: Specify Filter* (Fast Path **43**).

Uploading Large Dumps From a Standalone Dump Tape

Large dumps (that is, dumps of partitions that are larger than approximately 1 GB) cannot be uploaded from the dump tape into z/VSE using the INFOANA ONLOAD utility. This is because for such large dumps, the INFOANA utility cannot obtain enough storage below the 16 MB line. The job will end with these return codes:

```
BLN9002I ERROR IN EXTERNAL ROUTINE, RETURN CODE = 12, REASON CODE = 712
BLN3002I ONLOAD FAILED, REASON CODE = 3012
```

To avoid this problem, you can use a DITTO job (shown in Figure 10 on page 18) to upload the dump file into the z/VSE dump library. In this job, "file 6" represents the partition dump.

```
// JOB TEST
* MOVE DUMP FILE FROM TAPE TO DUMP LIBRARY
*
// UPSI 1
// PAUSE - PLEASE MOUNT SCRATCH TAPE ON 181
// EXEC DITTO
$$DITTO REW OUTPUT=181
$$DITTO TL INPUT=181,INFILE=6,RECFMIN=FB,LIBOUT=SYSDUMP.BG,
$$DITTO    MEMBEROUT=DBG00001.DUMP,RECFMOUT=S
/*
/&
```

Figure 10. Sample Job: Upload a Large Dump into a z/VSE Dump Library

If the dump was taken on a *z/VM system*, you can upload the dump file into the z/VM dump library using these CMS statements:

```
TAPE REW
TAPE FSF 5
FILEDEF OUT DISK dump_name dump_type mode ( RECFM F LRECL 4112 BLKSIZE 4112
FILEDEF IN  TAP1  (RECFM F LRECL 4112 BLKSIZE 4112
MOVEFILE IN OUT
```

Note:

1. In the above statements, File 6 (forward tape file 5) is the partition dump on the tape.
2. Loading of Memory Object dumps is limited to dumps smaller than 2 GB.

If required, you can now transfer the dump from the z/VSE or z/VM dump library to IBM Support (for further analysis).

Chapter 4. Requesting a Dump

VSE/Advanced Functions offers various functions with which storage areas can be dumped. These functions differ in their output contents, output device, and way of activation. You may use these functions to isolate system program or application program errors.

Overview of Dump Requests

Table 1 on page 21 summarizes the dump functions offered by VSE/Advanced Functions. The table may help you to find the dump request function which is the most effective one for your particular error situation.

Table 1. Dump Requesting Functions

Initiated by/via	**Output Contents**	**Output Device**	**Requesting Function**
System (ABEND)	System, Part., Data Space Dmp	Dump Sublib. or SYSLST	OPTIONS to Request the Dump
Operator (Console)	System, Part., Data Space Dmp	Dump Sublib. or SYSLST	CANCEL Command
Operator (Console)	Selected Storage Areas	Tape or Printer	DUMP Command
Operator (Console)	System Storage®	Tape or Disk	STAND-ALONE DUMP Program
Programm./Oper. (Defined Event)	Selected Storage Areas	Tape, Printer or Buffer	SDAID Dump Trace
Programmer (Macro)	Macro Dependent	Macro Dependent	MACROS (PDUMP, DUMP, JDUMP, SDUMP, SDUMPX)

Note that the base structure of the ABEND dump, the DUMP command dump, and the stand-alone dump is shown under "Dump Contents Overview" on page 3.

Each of the dump requesting functions listed in Table 1 on page 21 is described in the following sections.

Options to Control the ABEND Dump

The ABEND dump function is internally called when the VSE/Advanced Functions system detects an ABEND condition or when a CANCEL command has been given (see the following section).

Using the job control options shown below you can define whether you want to suppress a dump, or which kind of dump you want to take, and whether you want the dump to be stored in a dump sublibrary or printed on a particular output device.

Use the STDOPT command or statement to specify options for **all** jobs in the system. This must be entered in the BG partition, but it affects all partitions. Use the // OPTION statement to override these options for one job.

For a detailed description of the OPTION statement, refer to the z/VSE System Control Statements.

Options to Control the Dump Contents

The options controlling the **dump contents** can be set with the STDOPT or the OPTION statement.

STDOPT DUMP=YES
Requests a system dump.

STDOPT DUMP=PART
Requests a partition dump.

STDOPT DUMP=NO
Suppresses the ABEND dump.

STDOPT DSPDUMP=YES
Requests a data space dump.

STDOPT MODUMP=YES
Requests a memory object dump.

OPTION DUMP
Requests a system dump.

OPTION PARTDUMP
Requests a partition dump.

OPTION NODUMP
Suppresses ABEND dump.

OPTION DSPDUMP
Requests a data space dump.

OPTION MODUMP
Requests a memory object dump.

Note: You will not get any dump output if you use the STXIT PC or STXIT AB macro, even if you include the DUMP or PARTDUMP option.

Options to Control the Output Destination

The options controlling the **output destination** can be set with the STDOPT or the OPTION statement.

STDOPT SYSDUMP=NO
Dump to SYSLST

STDOPT SYSDUMP=YES
Dump to Library

OPTION NOSYSDUMP
Dump to SYSLST

OPTION SYSDUMP
Dump to Library

The output of the ABEND dump is either written into the dump sublibrary for the partition or it is printed on SYSLST.

The ABEND dump function writes the output to a ***dump sublibrary*** if the:

- **Dump Library**(named SYSDUMP) and appropriate **sublibrary** has been created.
- **LIBDEF statement**for the dump sublibrary has been submitted (usually during the ASI procedure for the partition).
- **Job Control option**

  STDOPT SYSDUMP=YES or OPTION SYSDUMP has been specified.
- Associated dump library is **not full**.

If one of the above is not true the dump is printed on ***SYSLST***.

Note: The dump is lost if it cannot go to SYSDUMP, and SYSLST has not been assigned. Also, the output of the ABEND dump routine is suppressed if SYSLST is assigned to a CKD-type disk device.

The contents of a system or a partition dump are described under "Contents of the ABEND Dump Output" on page 5.

How the SYSDUMP library can be defined is described under "Establishing the Dump Sublibraries" on page 12.

For a description of how to print the ABEND dump from a dump sublibrary see "Printing Dump Information" on page 182.

Requesting a Dump by the CANCEL Command

The CANCEL command, when used as a job control command cancels the execution of the current job in the partition in which the command is given. No dump is produced by the CANCEL job control command.

A detailed description of the options for the CANCEL command is given in the publication z/VSE System Control Statements. How to print a CANCEL command dump (ABEND dump) from a dump sublibrary is described under "Printing Dump Information" on page 182.

Requesting a Dump by the DUMP Command

The DUMP command causes selected areas of virtual address space or data space storage to be dumped.

A detailed description of the options for the DUMP command is given in the publication z/VSE System Control Statements. How to print the DUMP command output from tape is described under "Printing a Dump Stored on Tape or Disk" on page 202.

Taking a Stand-Alone Dump

The following steps describe how to invoke the dump process using the Standalone Dump Program. Note, however, that the procedure outlined below is only a generalized description of the dump process. For detailed information on the actual steps to be performed please consult the appropriate publication of your processor.

CAUTION: Do not reset (clear) the processor storage before taking the dump.

1. Do a STORE STATUS.

   Note: If your z/VSE system runs under z/VM®, you must first issue the CP SET RUN OFF command and then the CP STORE STATUS command.

   With the STORE STATUS step you save machine information that would otherwise be lost. This information is essential for error diagnosis.
2. Record the contents of low-address storage bytes X'00' to X'17'. Use the hardware DISPLY/ALTER function outlined under "Hardware Alter/Display" on page 228. To interpret the data stored in these bytes refer to "VSE/Advanced Functions Codes and SVC Errors" in the z/VSE Messages and Codes Volume 1 manual.
3. Mount a stand-alone dump tape (if the output is to be written on tape).
4. IPL the stand-alone dump tape or disk.

   Note: SADUMP supports the IPL load parameter. It may be used to specify the preferred communication device.

   To determine the communication path, SADUMP first analyzes the load parameter. If the hardware does not support the load parameter, selection of the communication device is determined during the creation of the dump program by the DOSVSDMP utility.

CAUTION: Do not reset (clear) the processor storage at this point.

The system now takes a stand-alone dump. The following message will be issued:

```
4G34I z/VSE STANDALONE DUMP IN PROGRESS ON TAPE cuu | DISK cuu
```

The following message indicates the end of the dump operation:

```
4G10I STANDALONE DUMP COMPLETE
```

If a problem occurred during processing, the following message is issued:

```
4G35I PROBLEM ENCOUNTERED DURING SA DUMP PROCESSING. REASON CODE nnnn
```

If the dump is on tape or on a work disk, the system enters a hard wait at dump completion. If the dump is on SYSRES, VSE is re-IPLed.

You need not regenerate the Standalone Dump Program after it has been used. The dump program remains useable for all subsequent stand-alone dump requests.

Note: An installation using only SCSI disks requires that the stand-alone dump program is generated on tape. Stand-alone dump processing does *not* work on SCSI disks!

Incorrect information in the system may result in only a dump of processor storage being taken. The Standalone Dump Program collects only those pages which are in processor storage at that moment, without address translation. Possible causes are, among others:

- Low core overlaid
- SYSCOM overlaid
- Page or segment tables not available or invalid.

Incorrect information in the system may also prevent the program from issuing messages on SYSLOG.

The output of the Standalone Dump Program can be written on more than one dump tape. At end-of-volume the stand-alone dump tape will be rewound and unloaded. An information message will be issued to the console:

```
4G36I END OF VOLUME ON DUMP TAPE cuu. MOUNT NEW TAPE OR RE-IPL VSE
```

The Standalone Dump Program will not wait for a reply. As soon as the new tape becomes ready, the dump will continue. If the operator decides to terminate stand-alone dump processing, he just re-IPLs VSE.

This multiple-tape support also allows stand-alone dump processing to continue if a tape error occurs in the middle of a tape. The operator will receive the following message:

```
4G37I ERROR ON DUMP TAPE cuu. MOUNT NEW TAPE OR RE-IPL VSE
```

The Standalone Dump Program and its output are described under "The Stand-Alone Dump (SADUMP) Program" on page 7.

The creation of the Standalone Dump Program is described under "Creating the Standalone Dump Program" on page 27.

A description of how to print the stand-alone dump can be found under "Printing a Stand-Alone Dump with Info/Analysis" on page 204.

Requesting a Dump on Event (SDAID Dump)

You may define that a dump has to be produced whenever a certain trace event occurs. The OUTPUT definition of the SDAID program is used for this purpose.

The following SDAID specifications for dump areas are possible:

- Partition
- Phase
- Area specified by storage addresses
- Area addressed by a register
- Area addressed by a pointer
- Control blocks or tables addressed by name.

The SDAID program is fully described in Chapter 7, "SDAID Overview," on page 45.

Requesting a Dump from a Program

```
DUMP Macro,  JDUMP Macro,   PDUMP Macro,   SDUMP Macro,   SDUMPX Macro
```

VSE supports the requesting of address space, data space, or memory object dumps through dump macros. These macros may be issued in any program written in assembler language.

If your program issues the macros DUMP or JDUMP, VSE/Advanced Functions terminates task processing and dumps the contents of the entire supervisor plus the used part of the system GETVIS area, or, if the options DUMP=NO (NODUMP) or DUMP=PART (PARTDUMP) are active, some supervisor control blocks plus the registers and the contents of the partition that issued the macro.

The PDUMP macro provides a dump of the general registers and of the storage area you defined with the macro operands on SYSLST regardless of the active options. Note however, if SYSLST is assigned to a CKD-type disk device, no output will be produced.

Detailed information on the output device and the output contents of the dump macros and the STXIT macro, are given in the z/VSE System Macros Reference.

Printing the Stored Dump

To print dumps stored on tape/disk or in a partition's dump sublibrary use Info/Analysis.

For information on Info/Analysis refer to “Info/Analysis” on page 10.

Archiving Expired or Unrequired Dumps

You may place dumps in the dump archives that is provided by the *Storage Dump Management* dialog.

You can also use REXX procedure DMPMGR to regularly delete expired dumps or dumps that are no longer required.

For details of both these facilities, refer to the publication z/VSE Guide for Solving Problems.

Chapter 5. The DOSVSDMP Utility

This topic describes the functions of the DOSVSDMP utility that is used in problem determination.

The DOSVSDMP Utility Functions

The DOSVSDMP utility is used to create the Standalone Dump Program with which virtual storage can be dumped. The utility can also be used to print the output of the DUMP command, the Standalone Dump Program (from tape or disk), the SDAID program, and IPL diagnostic information.

Run the DOSVSDMP utility in a partition with at least 192K of virtual storage.

The DOSVSDMP utility includes the following functions:

- "Creating the Standalone Dump Program" on page 27 (see below)
- "Dumps Printed with DOSVSDMP" on page 30, which describes how a DUMP command dump or a stand-alone dump can be printed.
- "Printing an SDAID or DUMP Command Produced Tape" on page 31.

Functions of the DOSVSDMP Utility

The DOSVSDMP utility includes the following functions:

- "Creating the Standalone Dump Program" on page 27 (see below)
- "Dumps Printed with DOSVSDMP" on page 30, which describes how a DUMP command dump or a stand-alone dump can be printed.
- "Printing an SDAID or DUMP Command Produced Tape" on page 31.

Creating the Standalone Dump Program

The Standalone Dump Program is mainly used in case of a hard or soft wait or if a system loop occurred. You can generate the Standalone Dump Program to reside on magnetic tape or disk (a z/VSE-based SCSI disk or z/VSE-based virtual disk are not valid as program residence.)

It is recommended to create the Standalone Dump Program on tape or on a work disk. If you create the Standalone Dump Program on your SYSRES disk, then any IPL request first causes a stand-alone dump to be taken. When the dump program has completed execution, it transfers control to the IPL program. If a dump is not needed, you can avoid the time consuming stand-alone dump processing by selecting the option CLEAR on the program load panel. The option CLEAR defines a fast path through the Standalone Dump Program which will immediately transfer control to the IPL program of z/VSE.

If you create the Standalone Dump Program on disk, two data sets (IJSYSDI and IJSYSDU) are required, as described under "Dump Program File and Dump Data Set" on page 29.

For processing the dump see "Printing a Dump Stored on Tape or Disk" on page 202.

To generate a Standalone Dump Program, invoke DOSVSDMP by entering

```
// EXEC DOSVSDMP
```

The program, once it receives control, prompts you for further control information as shown in Figure 11 on page 28.

Prompt Message

```
4G01D SELECT ONE OF THE FOLLOWING FUNCTIONS:
1 CREATE STAND ALONE DUMP PROGRAM
2 SCAN DUMP TAPE/DISK
3 PRINT DUMP TAPE/DISK
4 PRINT SDAID TAPE
R END DOSVSDMP PROCESSING
```

Enter **1** to create a stand alone dump program on tape or disk. The DOSVSDMP utility responds with

Prompt Message

```
4G04D SPECIFY ADDRESS OF DUMP DEVICE (CUU OR SYSNNN)
```

The device defined with SYSNNN or CUU can be a tape or disk.

Note: Neither the utility DOSVSDMP nor the generated Standalone Dump Program supports streaming mode on tape devices.

If the specified device address is that of a disk unit, DOSVSDMP responds with

Prompt Message

```
4G02D CREATE THE STAND ALONE DUMP PROGRAM
1 ON A WORK DISK
2 ON A SYSRES DISK
R END DOSVSDMP PROCESSING
```

Note: The Standalone Dump Program *cannot* be located on an FBA-SCSI disk!

Figure 11. Sample: Standalone Dump Program Generation

Enter **1** if you want to create the stand-alone program on a (non-SYSRES) work disk. In this case DOSVSDMP creates a VTOC entry for a dump program file IJSYSDI, for which you have to specify labels (see "Dump Program File and Dump Data Set" on page 29).

Enter **2** if you want to create the stand-alone program on a SYSRES disk. In this case, no labels are required for IJSYSDI. DOSVSDMP creates the dump program within the disk extent reserved for the system library. Note, however, that if you create the Standalone Dump Program on the SYSRES disk, a new stand-alone dump is taken with every subsequent IPL (unless you specify CLEAR).

In both cases you have to specify labels for a dump data set IJSYSDU (see "Dump Program File and Dump Data Set" on page 29). You can remove the Standalone Dump Program from the system disk by entering option 3 (Remove Standalone Dump Program from a SYSRES disk) from the **Dump Program Utilities** panel of the Interactive Interface. The

Completion Message

```
4G09I DUMP PROGRAM HAS BEEN CREATED
```

indicates the successful generation of the dump program.

If the dump file is on disk, the completion message is followed by a message indicating the dump file capacity:

Capacity Message

```
4G27I DUMP FILE CAPACITY IS nnnnnnnn,nn K BYTES
```

Note: If the Standalone Dump Program was created on the DOSRES or SYSWK1 disk, you have to recreate it after indirect service application. This is because during service application, the Standalone Dump Program is overwritten by IPL records.

The description of how the Standalone Dump Program is executed can be found under "The Stand-Alone Dump (SADUMP) Program" on page 7.

Using the PARM Parameter: Instead of using the Menu selections described previously, you can simply use the **PARM** parameter together with EXEC DOSVSDMP. Here is an example:

```
 // EXEC DOSVSDMP,PARM='CREATE DUMP DEVICE=cuu'
```

Dump Program File and Dump Data Set

Two data sets are required to create a Standalone Dump Program on a disk pack: the dump program file and the dump data set. These files have to be defined on the same disk pack.

Dump Program File (IJSYSDI)

If the dump program is to be created on a SYSRES disk, the dump program becomes part of the system library and you need not specify labels for the dump program file. If the dump program is to be created on a non-SYSRES disk, you have to define the required disk space explicitly and create the following labels for IJSYSDI:

```
// DLBL IJSYSDI,'VSE.DUMP.PROGRM'
// EXTENT ,,,,1,7    (for CKD)
// EXTENT ,,,,2,128  (for FBA but not SCSI)
```

Be aware of the following:

- SCSI disks can be used via the VM-Emulated FBA Support. Then they appear to z/VSE as 9336 Model 20 FBA disks.
- SCSI disks can be used as z/VSE FCP-attached SCSI disk support. Then the restrictions for SCSI disks take place.

The dump program occupies the first eight tracks of a CKD disk or the first 130 blocks on an FBA disk (but *not* SCSI). Track 0 of a CKD disk and blocks 0 and 1 of an FBA disk are used for IPL records.

Dump Data Set (IJSYSDU)

The dump data set may be defined anywhere on the disk pack. Labels for IJSYSDU are required for Standalone Dump Program creation, for printing or scanning the dump data set, and for the dump onload function:

```
// DLBL IJSYSDU,'VSE.DUMP.FILE'
// EXTENT ,,,,rel-track,no-of-tracks    (for CKD)
// EXTENT ,,,,block,no-of-blocks        (for FBA but not SCSI)
```

You need to define enough space to dump the supervisor, the shared virtual area (SVA), and space for those partitions and/or data spaces that you want to dump. If there is not sufficient space, the areas will be dumped until the space is full.

To make sure that the dump data set is large enough, calculate the amount of storage you want to have dumped, add 5% to the result, and compare it to the size provided by message 4G27I. If the size is too small, increase it and rerun the job.

If the dump data set is too small to contain a complete stand-alone dump, the remainder of the dump is dropped. The dump data set will contain only one stand-alone dump at a time. Any subsequent dump will overwrite the previous dump. ABEND dumps or attention routine dumps cannot be written into the dump data set.

Scanning the Dump Files on Disk or Tape

The SCAN function of DOSVSDMP provides a file directory of the dump tape or disk. After having invoked DOSVSDMP by entering:

```
// EXEC DOSVSDMP
```

the program prompts you for further information as shown below.

Prompt Message

```
 4G01D SELECT ONE OF THE FOLLOWING FUNCTIONS:
 1 CREATE STAND ALONE DUMP PROGRAM
 2 SCAN DUMP TAPE/DISK
 3 PRINT DUMP TAPE/DISK
 4 PRINT SDAID TAPE
 R END DOSVSDMP PROCESSING
```

Enter **2** to scan the dump tape or the dump data set on disk. DOSVSDMP prints the following information on SYSLST:

```
1. For SCAN DUMP DISK:

PRINTOUT OF VSE DUMP DATA SET
DIRECTORY OF VSE DUMP DATA SET
DUMP FILE  DUMP TYPE  NAME      DATE        DATA DUMPED
---------  ---------  --------  ----------  --------------------------
   001     SADUMP               2012/10/22  SUPERVISOR+SVA
   002     SADUMP               2012/10/22  PMRAS-R
   003     SADUMP               2012/10/22  PMRAS-00
   004     SADUMP     SECSERV   2012/10/22  FB-PARTITION
   005     SADUMP     VTAMSTRT  2012/10/22  F3-PARTITION
   006     SADUMP     CICSICCF  2012/10/22  F2-PARTITION
   007     SADUMP     POWSTART  2012/10/22  F1-PARTITION
END OF DUMP

2. For SCAN DUMP TAPE:

DIRECTORY OF VSE DUMP TAPE
DUMP FILE  DUMP TYPE  NAME      DATE       DATA DUMPED
---------  ---------  --------  --------   ---------------------
  001                                      DOES NOT CONTAIN DUMP DATA
  002                                      DOES NOT CONTAIN DUMP DATA
  003      SADUMP                          SUPERVISOR+SVA
  004      SADUMP                          PMRAS-R
  005      SADUMP                          PMRAS-00
  006      SADUMP     NO-NAME              BG-PARTITION
  007      SADUMP     SECSERV              FB-PARTITION
  008      SADUMP     NO-NAME              FA-PARTITION
  009      SADUMP     NO-NAME              F9-PARTITION
  010      SADUMP     PAUSEF8              F8-PARTITION
  011      SADUMP     NO-NAME              F7-PARTITION
```

Figure 12. Sample: Directory of Dump Disk/Tape

Using the PARM Parameter: Instead of using the Menu selections described previously, you can simply use the **PARM** parameter together with EXEC DOSVSDMP. Here is an example:

```
 // EXEC DOSVSDMP,PARM='SCAN DEVICE=cuu'
```

Dumps Printed with DOSVSDMP

How to print the dumps produced by the Standalone Dump Program and the DUMP command in unformatted form is discussed in this section. Normally Info/Analysis is used to process and print dump tapes. In exceptional cases the use of the DOSVSDMP utility may be necessary, for example:

- If none of your dump sublibraries are big enough to hold the stand-alone dump;
- If the dump was taken with the DUMP BUFFER,cuu command.

The printed output of the DOSVSDMP utility contains for both DUMP command tape or stand-alone dump tape/disk, the following:

- Symptom record.
- Unformatted dump data.

Sample DOSVSDMP Print Setup

To print a dump from tape or disk using the DOSVSDMP utility, invoke DOSVSDMP by submitting the control statements shown in Figure 13 on page 31.

Figure 13. Sample: Dump Tape Printed with DOSVSDMP

The utility prompts you by messages for further control information, which you enter at SYSLOG.

// JOB DOSVSDMP
// EXEC DOSVSDMP

DOSVSDMP prompts you by messages at SYSLOG to define the operation you want to perform, with:

Prompt Message

```
   4G01D SELECT ONE OF THE FOLLOWING FUNCTIONS:
   1 CREATE STAND ALONE DUMP PROGRAM
   2 SCAN DUMP TAPE/DISK
   3 PRINT DUMP TAPE/DISK
   4 PRINT SDAID TAPE
   R END DOSVSDMP PROCESSING
```

Enter **3** to invoke DOSVSDMP Print Dump Tape/Disk processing.

The DOSVSDMP utility response is:

Prompt Message

```
   4G04D SPECIFY ADDRESS OF DUMP DEVICE (CUU OR SYSNNN)
```

Enter 280, for example, if the dump tape is mounted on the tape drive 280.

If you have selected option 3, DOSVSDMP also prompts you for the number of the dump file that you want to print. (Option 2 - SCAN DUMP TAPE/DISK - gives you a directory of the dump files on the dump.

See also "Scanning the Dump Files on Disk or Tape" on page 29).

Prompt Message

```
   4G30D SPECIFY FILE NUMBER
```

Enter 4, for example, if you want to print file 4.

Now the DOSVSDMP utility starts printing the dump on SYSLST.

After print completion, control is returned to Job Control.

Using the PARM Parameter Instead of using the Menu selections described previously, you can simply use the **PARM** parameter together with `EXEC DOSVSDMP`. Here is an example:

```
 // EXEC DOSVSDMP,PARM='PRINT DUMP DEVICE=cuu FILE=n'
```

Printing an SDAID or DUMP Command Produced Tape

You may specify that the SDAID trace information is to be recorded on tape. DOSVSDMP can be used to retrieve this information from tape and to print it on SYSLST. This is done by responding to DOSVSDMP prompts as shown below. Always use this option of DOSVSDMP to print dumps produced in response to the attention routine command

```
DUMP BUFFER,cuu
```

When the utility gets control, it prompts you for further definitions via SYSLOG, as shown in the example of Figure 14 on page 31.

Figure 14. Sample Job: Printing SDAID Tape with DOSVSDMP

```
// JOB SDAID
// EXEC DOSVSDMP

   DOSVSDMP prompts you to define the operation you want to perform:
```

Prompt Message

```
4G01D SELECT ONE OF THE FOLLOWING FUNCTIONS:
1 CREATE STAND ALONE DUMP PROGRAM
2 SCAN DUMP TAPE/DISK
3 PRINT DUMP TAPE/DISK
4 PRINT SDAID TAPE
R END DOSVSDMP PROCESSING
```

Enter **4** to invoke DOSVSDMP Print SDAID Tape processing.

The DOSVSDMP utility responds with:

Prompt Message

```
4G05D SPECIFY ADDRESS OF SDAID TAPE (CUU OR SYSNNN)
```

Enter 280, for example, if the SDAID output tape is mounted on the device 280.

The DOSVSDMP utility now responds with:

Prompt Message

```
4G30D SPECIFY FILE NUMBER
```

```
Enter 2, for example, if the second file contains the SDAID output
you want to print.
The file number is determined by the number of STOPSD commands
given in the SDAID session.  (Every STOPSD command writes a tapemark
on the tape if there was any trace event.)
If, for example, you issue three times STARTSD/STOPSD within an
SDAID session, you get three trace files on your trace output tape.

DOSVSDMP prints the tape on the device assigned to SYSLST.
After print completion, control is returned to Job Control.
```

Using the PARM Parameter: Instead of using the Menu selections described previously, you can simply use the **PARM** parameter together with EXEC `DOSVSDMP`. Here is an example:

```
// EXEC DOSVSDMP,PARM='PRINT SDAID TAPE=cuu FILE=n'
```

Part 2. Interactive Trace Program

Chapter 6. Interactive Trace Program

Introduction

The interactive trace program is the tracing tool for z/VSE application programs. It traces the execution of application programs running in static or dynamic partitions. The interactive trace program is activated via the // EXEC statement and controlled interactively from the z/VSE master console or from a user console. It operates at the level of machine instructions and virtual storage addresses, similar to the CP debugging facilities in z/VM.

The interactive trace program provides the following traces:

- Branch trace
- Instruction trace
- Storage alteration trace
- ABEND trace.

Branch Trace

The branch trace monitors branch instructions. The trace program displays all branch instructions which transfer control to an address which is located within a specified storage area. That means, branches are only recorded if the **target** address of the branch is located within the specified address range.

Instruction Trace

The instruction trace monitors the instructions executed within a specified storage area. An instruction is traced if the first byte of the instruction is contained in the specified storage area. The trace program displays also EXECUTE instructions if the first byte of the target of an EXECUTE is within the designated storage area.

Storage Alteration Trace

The storage alteration trace monitors storage alterations within a specified storage area. A storage alteration event occurs even if the value stored is the same as the original value. However, monitoring does not apply if data is altered by a channel program or by system control programs.

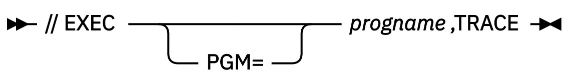
ABEND Trace

The ABEND "abnormal end" trace allows interactive debugging if a user program terminates abnormally. In case of an ABEND, the termination routines display the cancelation message on the screen and transfer control to the console operator. The operator can inspect storage data or register contents to determine the cause of the cancelation. It is, however, not possible to change the program status and return to normal operation via the GO command (see "GO Command" on page 38) with a branch address. The task termination is already in progress at that time. The GO command can only be used to resume the termination process. It is also possible to modify the dump option to DUMP, PARTDUMP, or NODUMP.

The ABEND trace does not become active if an AB exit routine (STXIT routine) with the options EARLY or NODUMP is defined. In these cases control is transferred to the user exit routine before the trace is invoked. The code of the exit routine can, however, be traced via an instruction trace if the code segment of the exit routine is defined as tracing range.

If the branch trace, the instruction trace, or the storage alteration trace display an instruction on the screen, this instruction has already been executed. If the trace displays an interruptible instruction, like an MVCL, the PSW, the general registers, and storage data at the time of the interruption are displayed. If the MVCL is partly processed, the PSW still points to the MVCL instruction. If the execution of the MVCL is completed, the PSW points to the instruction after the MVCL instruction. If the trace program displays an SVC instruction, the related supervisor service has not been started yet.

Trace Activation



You start the interactive trace program with the parameter TRACE in the EXEC statement. An example of the trace initialization is shown in Figure 15 on page 40. The invoked trace function is active for the duration of one VSE job step.

The parameter TRACE implicitly defines an instruction trace and an ABEND trace. These trace definitions allow the console operator to get interactive control over the program to be traced. The instruction trace passes control to the console operator at the beginning of a user program, the ABEND trace allows debugging when a program terminates abnormally.

The instruction trace defined implicitly via the TRACE parameter traces all instructions executed within the partition. Trace boundaries are the partition begin and end address. The user program stops after the first instruction has been executed. The trace program displays the interrupted instruction (preceded by a reply identification) and waits for an operator response. The operator answers with an interactive trace command. The operator may use the implicitly defined traces to step through all instructions of the program, or replace these implicitly defined traces by specific trace definitions. The implicitly defined traces remain in effect until they are explicitly deleted by the operator.

Interactive Trace Commands

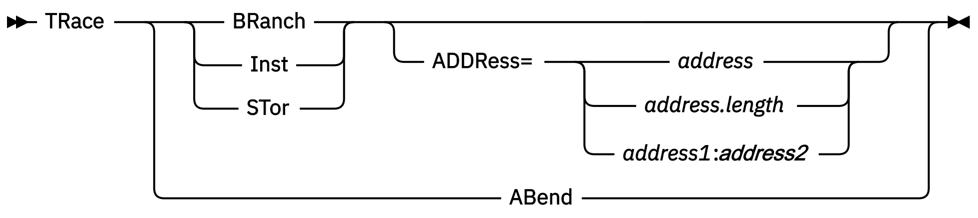
This topic describes the following interactive trace commands:

- TRACE
- QUERY
- DISPLAY
- ALTER
- GO

Most of these commands may be abbreviated. The possible abbreviation is shown through lowercase letters. An example of an abbreviated command is:

```
tr i addr=4037ac.1c
```

TRACE Command



The TRACE command defines the type of trace to be activated (branch trace, instruction trace, storage alteration trace, or ABEND trace). It is possible to issue up to 100 trace definition statements within one interactive tracing session.

The parameter ADDRESS defines the tracing range. It has different meanings for the different types of traces.

Branch trace:
The parameter ADDRess= defines the branch target area.

Instruction trace:
The parameter ADDRess= defines the storage area within which instruction execution is monitored. The address range must be part of the partition area. A tracing area outside the user partition is rejected. If the specified tracing range crosses partition boundaries, only the range located within the partition is accepted.

Storage alteration trace:
The parameter ADDRess= defines the area within which storage alteration is to be monitored.

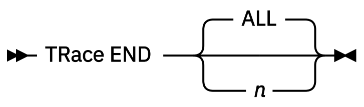
ABEND trace:
The parameter ADDRess= is not applicable for the ABEND trace.

If the parameter ADDRESS is omitted, the whole user partition is assumed as tracing range. The following examples explain the different forms of the address parameter.

`ADDRESS=410C1F`
specifies a one-byte storage interval.

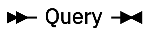
`ADDRESS=460C1F.C`
specifies a storage interval of 12 bytes.

`ADDRESS=40031C:400328`
specifies a storage interval by its virtual start address and end address.



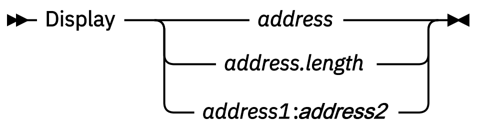
The TRACE END command deletes one or all traces specified for a partition. The parameter '*n*' addresses a trace statement by its trace identification (obtained via the QUERY command). The keyword ALL (default) deletes all traces specified for a partition.

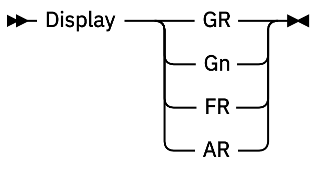
QUERY Command

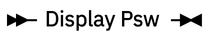


The QUERY command displays a list of all traces active for a user partition. The displayed trace-identification may be used in a subsequent TRACE END command to delete one of the specified traces. An example of the QUERY command is shown in Figure 16 on page 40.

DISPLAY Command







The DISPLAY command displays either storage data, or the general purpose registers (GR, Gn), the floating point registers (FR), the access registers (AR), or the Program Status Word (PSW). The specification `DISPLAY GR` displays **all** general purpose registers, the specification `DISPLAY Gn`, displays a particular general purpose register. An example of the DISPLAY command is shown in Figure 18 on page 42.

ALTER Command

►► Alter *address* DATA= *data* ►◄

►► Alter Gn DATA= *data* ►◄

The ALTER command allows to alter storage data or the contents of a general purpose register.

- **Altering storage data**: The *address* parameter denotes the storage address where data is to be altered. The DATA parameter describes the new storage data by its hexadecimal representation. Any two hexadecimal digits describe the contents of one byte in storage. The specified data is not padded; that means, it is required to enter an even number of hexadecimal digits. It is possible to enter up to 16 hexadecimal digits in order to alter up to 8 bytes. It is not possible to alter storage locations outside the user partition, or to alter the mask portion of the stored PSW. Example:

  The specification

  ```
  ALTER 400312 DATA=03FEC7
  ```

  alters the contents of addresses 400312, 400313, 400314 to the values 03, FE, C7 respectively.
- **Altering the contents of a general purpose register**: The parameter Gn denotes the general purpose register *n*. The DATA parameter describes the new contents of the specified general purpose register. The entered data is padded on the left with binary zeros. It is possible to enter up to 8 hexadecimal digits.

  Example:

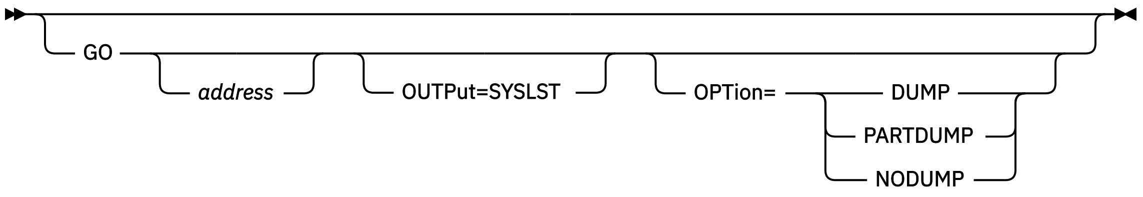
  The specification of

  ```
  ALTER G5 DATA=12C
  ```

  enters the value of 0000012C into general purpose register 5.

(Another example of the ALTER command is shown in Figure 18 on page 42.)

GO Command



The GO command reactivates the stopped user program. The program continues processing at the specified address. This address is not checked for validity. If the address parameter is omitted, the program continues processing with the next sequential instruction. The interactive trace command ignores the specified branch address if the last displayed instruction is an SVC instruction, or if a cancel condition has occurred and the termination routines have already issued the termination messages. In this case the GO command has the only effect to resume the termination process.

The parameter OUTPut=SYSLST switches the tracing mode from interactive tracing to batch tracing. The trace output lines are printed on a printer device or they are written into the VSE/POWER list queue. Switching to batch mode is not possible if the logical unit SYSLST is unassigned or assigned to a tape or disk device.

The parameter OPTion= modifies the temporary dump options. In case of an abnormal termination, the dump routines will print either a full dump (DUMP), a partition dump (PARTDUMP), or no dump at all (NODUMP). If the parameter OPTION is omitted, the dump options remain unchanged.

If the parameters *address*, OUTPUT, and OPTION are omitted, the command name GO can be omitted, too. A reply-ID without any parameter is processed as a 'GO' command.

Tracing in a User Partition with Subtasks Attached

The interactive trace program traces the main task and all attached subtasks. If two tasks execute in the specified tracing range concurrently, both tasks are traced. Different VSE users can activate the trace program independently in different partitions. It is possible that several partitions present trace messages at the same time. Within one partition, however, only one task can present a tracing event at a time. The first task arriving at a specified tracing event locks the tracing routine for exclusive usage. The interrupted task writes a console message and waits for an operator response. The other tasks continue processing until they arrive at a tracing point. As long as the operator issues interactive trace commands, no other task can issue a tracing message. When the operator resumes program operation (via a GO command), all tasks with a pending tracing message will be activated. The subtask with the highest priority will present its tracing message at the screen.

Scope of Tracing

The interactive trace program is designed to trace user programs. It cannot be used to trace supervisor routines, Job Control statements, or attention routine commands. These restrictions have been introduced to keep the impact of tracing on the operations in other partitions to a minimum. The system routines often lock non-reentrant system resources. Any other task in the system competing for the same resource might enter a wait state until the locked resource becomes free. Therefore it is not tolerable to interrupt a system routine for interactive tracing.

A program routine which runs in a user partition and owns the Logical Transient Area (LTA), cannot be traced interactively. An example of such a user routine is an OPEN exit routine which is called from the LTA via a CALL/RETURN interface. The trace program does not stop for exit routine instructions. Programs running in other partitions might wait for the LTA to become free. It is, however, possible to trace such a routine in batch mode. You may use the GO command with the parameter OUTPut=SYSLST to trace all or selected instructions of the exit routine on SYSLST.

In exceptional cases it is possible that an interactive trace of a user written routine may have an impact on the performance of programs running in other partitions. For example, such an interference with other partitions may occur if the program uses the LOCK macro or the track-hold option of a DTF macro to synchronize processing with programs in other partitions.

Restrictions for Programs Using the PER Function

The interactive trace program uses the Program Event Recording (PER) function of the z/Architecture®. It is possible to run the interactive trace program in several partitions at the same time. However, it is not possible to run the interactive trace program concurrently with another program which uses the PER function.

The Interactive Trace Program versus SDAID

Some trace types of the SDAID program use the Program Event Recording function. These trace types are the branch trace, the instruction trace and the storage alteration trace. They **cannot** run concurrent with the interactive trace program. The SDAID initialization routine checks whether an interactive trace for any partition is already active. The SDAID STARTSD command is rejected if the interactive trace program is active for any partition. (For a description of the STARTSD command see Chapter 12, “Start/Stop and End the Trace,” on page 163.)

An SDAID session which does not use the PER function can run concurrently with the interactive trace program. These trace types do not interfere with the interactive trace program:

- CANCEL
- EXTERNAL
- GETVIS
- IO
- LOCK
- MON
- PGML
- PGMC
- SSCH
- SVC
- VTAMBU
- VTAMIO
- XPCC

Examples of the Interactive Trace Program

Trace Initialization Example

```
// JOB ABC
// EXEC PROG1,TRACE
/&

BG 0000 4I01I TRACE STARTED FOR PROGRAM PROG1
BG-0000 00600078  BALR  05C0   CC 0
```

Figure 15. Trace Example: Trace Initialization

Figure 15 on page 40 shows a job stream which initializes an interactive trace session for program PROG1.

The last two lines show the system response. Message 4I01I indicates that trace initialization was successful. The traced program stops its execution after the first instruction has been executed. The trace program displays the first instruction on the screen and waits for an operator response. The operator may now use the implicitly defined instruction trace to step through all instructions of the program, or replace this instruction trace by specific trace definitions.

Figure 16 on page 40 shows the available commands to modify the trace environment.

TRACE, TRACE END and QUERY Command Example

```
BG+0000 00400078  BALR  05C0   CC 0
0 query
BG 0000 001  TRACE  INST  ADDRESS=00600000:006AFFFF
BG-0000 002  TRACE  ABEND
0 trace end  1
BG-0000 4I09D SPECIFIED TRACE ENDED
0 trace inst address=403BA0.70
BG-0000 003  TRACE  INST  ADDRESS=00603BA0:00603C0F
0 trace stor address=4002ad
BG-0000 004  TRACE  STOR  ADDRESS=006002AD:006002AD
0 trace inst address=4017cc:4017ff
BG-0000 005  TRACE  INST  ADDRESS=006017CC:006017FF
0 query
BG 0000 002  TRACE  ABEND
BG 0000 003  TRACE  INST  ADDRESS=00403BA0:00403C0F
BG 0000 004  TRACE  STOR  ADDRESS=004002AD:004002AD
BG-0000 005  TRACE  INST  ADDRESS=004017CC:004017FF
```

Figure 16. Trace Example: TRACE, TRACE END and QUERY Command

Figure 16 on page 40 shows how a tracing environment can be modified. The operator deletes the implicitly specified instruction trace (001) and defines two new instruction traces (003 and 005) and one storage alteration trace (004). Trace 002 continues unaltered.

Batch Trace Example

```
0 exec testtrac,trace
BG 0000 4I01I TRACE STARTED FOR PROGRAM TESTTRAC
BG-0000 00400078  BALR  0530        CC 0
0
BG-0000 0040007A  B     47103024 -> 0040009E  CC 0
0
BG-0000 0040009E  NOPR  0700        CC 0
0
BG-0000 004000A0  BAL   4510303E -> 004000B8  CC 0
0
BG-0000 004000B8  LR    1801        CC 0
0
BG-0000 004000BA  SVC   0A26        CC 0
0
BG-0063 004001A4  LA    41603313  = 0040038D  CC 0
63 go output=syslst
BG 0000 4I20I TRACING TERMINATED
BG-0000 1I00D  READY FOR COMMUNICATIONS.
```

```
***** START OF BATCH TRACE *****
0063 004001A8  STCM  BE67359F >> 00400619  CC 0
0063 004001AC  LA    41600020  = 00000020  CC 0
0063 004001B0  STC   426035A5 >> 0040061F  CC 0
0063 004001B4  L     58103606    00400680  CC 0
0063 004001B8  SVC   0A00        CC 3
0063 004001BA  L     58103606    00400680  CC 0
0063 004001BE  TM    91801002    004005DE  CC 0
0063 004001C2  BO    4710314E    004001C8  CC 0
0063 004001C6  SVC   0A07        CC 0
0000 004000BC  LTR   1211        CC 2
0000 004000BE  BM    47403104    0040017E  CC 2
0000 004000C2  NOPR  0700        CC 2
0000 004000C4  BAL   45103062 -> 004000DC  CC 2
0000 004000DC  LR    1801        CC 2
0000 004000DE  SVC   0A26        CC 2
0064 0040020C  LA    41603333  = 004003AD  CC 2
0064 00400210  STCM  BE6735AF >> 00400629  CC 2
0064 00400214  LA    41600020  = 00000020  CC 2
```

Figure 17. Trace Example: Batch Trace

Figure 17 on page 41 shows a program with sub tasks attached. The main task and the sub tasks have different reply identifications (0000, 0063, 0064). After the instruction at location 4001A4 has been executed, the operator issues the command `go output=syslst` to switch from the interactive tracing mode into the batch tracing mode. A fragment of the batch output on SYSLST is shown in the second part of the figure.

DISPLAY and ALTER Command Example

```
5 display psw
F5-0005 PSW = 470D0000 00400EE6
5 display gr
F5 0005 GPR   0 = 00000000 00400648 00003110 00400EF4
F5 0005 GPR   4 = 00403518 004001AE 004010E0 004000E0
F5 0005 GPR   8 = 00407000 004020E0 00000590 004030E0
F5-0005 GPR  12 = 00000006 00407894 804017FC 90000004
5 alter gc data=00000007
F5-0005 GPR  12 = 00000007
5 display 403017.20
F5 0005 00403010  00E1D8E0 00E1474E 00006FBC 00000644 *..Q....+..?.....*
F5 0005 00403020  00009DA0 00E263C8 00E18598 00FC4540 *.....S.H..eq....*
F5-0005 00403030  00004930 000094E8 00E623F8 00E26414 *......mY.W.8.S..*
5 display 403017
F5-0005 00403010  00E1D8E0 00E1474E 00006FBC 00000644 *..Q....+..?.....*
5 alter 403017 data=fefefe
F5-0005 00403010  00E1D8E0 00E147FE FEFE6FBC 00000644 *..Q.......?.....*
```

Figure 18. Trace Example: DISPLAY and ALTER Command

Part 3. SDAID Trace

In order to isolate a problem in a system or in an application program, you may need to know the exact sequence of execution steps which have been performed. To find out about the execution steps performed, you have to trace specific events. VSE offers a program which helps you in tracing specific events in your system. This program is called SDAID.

SDAID traces user and system programs running either below or above the 16MB line. The tracing range and the dump area are specified as 31-bit addresses.

You can initialize the SDAID traces by:

- Answers to prompts in the attention routine (AR);
- Direct input statements to the AR or a partition;
- Job control procedures.

This part of the publication describes each of the methods to initialize a trace and how the initialized trace is started/stopped or ended. Trace output examples are shown for each of the trace types.

If the SDAID program is new for you, read Chapter 7, “SDAID Overview,” on page 45 as an introduction.

If you are familiar with the SDAID program conventions, read the Summary in Table 3 on page 47. This Summary shows all trace types with references to their format descriptions for the various initialization methods.

Chapter 7. SDAID Overview

This topic gives an overview of the SDAID program and the various ways to initialize a trace; it includes considerations on the performance and the environment.

The SDAID Session

Basically you will do two (or three) things:

1. Initialize a trace.
2. Start, stop, or terminate the initialized trace.
3. Print trace data via DOSVSDMP (if trace output is on tape).

Interaction SDAID versus Interactive Trace Program

The SDAID branch trace, the instruction trace and the storage alteration trace use the Program Event Recording (PER) function of the z/Architecture. The interactive trace program also uses this hardware function. SDAID sessions which contain one of the above mentioned traces cannot be started if an interactive trace for any user partition is already active.

An SDAID session which does not use the PER function can run parallel to the interactive trace program. These trace types do **not** interfere with the interactive trace program:

- CANCEL
- EXTERNAL
- GETVIS
- IO
- LOCK
- MON
- OSAX
- PGMC
- PGML
- SSCH
- SVC
- VTAMBU
- VTAMIO
- XPCC

How to Initialize an SDAID Trace

You initialize a trace with mainly four statement types which have to be submitted to the SDAID program:

1. The SDAID statement to start the initialization process.
2. The OUTDEV specification to define the output device for the trace data.
3. The TRACE statements to define all necessary information for the trace, like the trace type and the area to be traced.
4. A statement which signals the end of the initialization process (/* or READY).

You submit the SDAID statements with one of the following methods:

- Direct input mode in the attention routine or partition.

- Job control procedures in a partition.
- Prompts in the attention routine (AR).

Initialization in Direct Input Mode

In direct input mode the SDAID information is entered in the form of commands to the attention routine or as SYSIN statements in a partition.

The SDAID program identifies the mode of initialization via the format of the TRACE and OUTDEV statement. In direct input mode these statements must contain at least one operand.

The following examples show two initialization jobs, one entered in a partition the other one entered via SYSIN.

Example of a trace initialization in direct input mode in the attention routine:

```
sdaid
AR 4C05I PROCESSING OF 'SDAID'   COMMAND SUCCESSFUL.
AR 1I40I READY
outdev tape=280
AR 4C05I PROCESSING OF 'OUTDEV'  COMMAND SUCCESSFUL.
AR 1I40I READY
trace ssch unit=009
AR 4C05I PROCESSING OF 'TRACE'   COMMAND SUCCESSFUL.
trace io unit=009 output=ccw
AR 4C05I PROCESSING OF 'TRACE'   COMMAND SUCCESSFUL.
ready
AR 4C05I PROCESSING OF 'READY'   COMMAND SUCCESSFUL.
AR 1I40I READY
```

Example of trace initialization by direct input mode statements read in from SYSIN:

```
// EXEC SDAID
OUTDEV TAPE=280
TRACE SSCH UNIT=009
TRACE IO UNIT=009 OUTPUT=CCW
/*
```

The direct input mode is described in Chapter 9, “Initialize an SDAID Trace in Direct Input Mode,” on page 83.

Initialization via Job Control Procedures

The easiest way to initialize a trace is to use catalogued procedures.

An example of such a trace procedure statement is shown below.

```
// EXEC PROC=SDIO,UNIT=009,TAPE=280
```

Initialization by procedures is described in Chapter 10, “Initialize an SDAID Trace via a Procedure,” on page 117.

Initialization via Prompts in the Attention Routine

You start the initialization process with the attention routine command ‘SDAID’. The necessary trace definitions are given in response to promptings after you entered the TRACE or OUTDEV statement without an operand.

You enter the prompt mode whenever you define these two commands without an operand. Example of a trace initialization via prompts in the attention routine:

```
sdaid ◊
AR 4C05I PROCESSING OF 'SDAID'  COMMAND SUCCESSFUL
outdev ◊
AR 4C08D SPECIFY OUTPUT DEVICE.+
tape ◊
  ...

◊ indicates the ENTER key pressed
```

Note that you enter the prompt mode also if you specify direct input mode statements combined with prompt mode statements like a question mark (? requests the help function of SDAID). The example below shows, how you can combine the two input modes. You would be prompted after the question mark has been processed.

```
TRACE SSCH AREA=BG ?
- direct input ───► | ◄─── prompt mode
```

The prompt input mode is described in Chapter 11, "Initialize a Trace in Prompt Input Mode," on page 133.

AR Commands to Start, Stop and End an Initialized Trace

You can start, stop, or end an initialized trace via attention routine (AR) commands. The table below shows an overview of these commands. A more detailed description about stopping, starting, and ending a trace is given in Chapter 12, "Start/Stop and End the Trace," on page 163.

Table 2. AR Commands to Start/Stop and End an Initialized Trace

STARTSD	Starts SDAID execution; may follow READY or STOPSD. **Note:** The old form of the command (STRTSD) is still accepted.
STOPSD	Suspends SDAID execution; allowed only after STARTSD.
ENDSD	Ends SDAID session; releases all system resources used by SDAID at any time.

Trace Type Summary

Find the trace type which you want to initialize in the following trace command summary. You will find references to the description of the trace type and to the format of the trace initialization statements for the various initialization methods. Locate the section according to the initialization method you choose.

The references to the available descriptions are under the following headings:

Des
Reference to the *description* of the trace type

Dir
Reference to the format description for initialization in *direct input mode*.

Prc
Reference to the format description for the initialization via *procedures*.

Prp
Reference to the description for the initialization in *prompt mode*.

Table 3. Trace Type Summary

Trace Type	Des	Dir	Prc	Prp
BRANCH* (provides trace of successfully executed branch instructions)	"BRANCH Trace" on page 56	"BRanch Trace" on page 89	"Branch Trace Initialization" on page 121	"BRanch Trace" on page 147

Table 3. Trace Type Summary (continued)

Trace Type	Des	Dir	Prc	Prp
BUFFER (provides trace of the trace buffer when it is full)	"BUFFER Trace" on page 57	"BUffer Trace" on page 89	-	"BUffer Trace" on page 147
CANCEL (provides trace of program - main task - cancel or EOJ)	"CANCEL Trace" on page 57	"CAncel Trace" on page 90	-	"CAncel Trace" on page 147
EXTERNAL (provides trace of external interrupts)	"EXTERNAL Trace" on page 57	"EXTernal Trace" on page 91	-	"EXTernal (External Interrupt) Trace" on page 147
GETVIS (provides trace of GETVIS / FREEVIS requests)	"GETVIS / FREEVIS Trace" on page 58	"GETVIS Trace" on page 92	-	"GETVis (Getvis / Freevis Request) Trace" on page 148
INSTRUCTION (provides trace of selected or all instruction(s) execution)	"INSTRUCTION Trace" on page 59	"INSTruction Trace" on page 94	"Instruction Trace" on page 123	"INSTruction (Instruction Execution) Trace" on page 149
IO (provides trace of I/O interrupts)	"IO Trace (I/O Interrupt)" on page 59	"I/O Interrupt Trace" on page 95	"SSCH and I/O Interrupt Trace" on page 124	"IO (I/O Interrupt) Trace" on page 149
LOCK (provides trace of LOCK / UNLOCK requests)	"LOCK / UNLOCK Trace" on page 60	"LOCK Trace" on page 96	-	"LOCK (Lock / Unlock of Resources) Trace" on page 150
MONITORCALL (provides trace of MC instructions)	"MONITORCALL Trace" on page 61	"MONitor Call Trace" on page 98	-	"MONitorcall Trace" on page 151
OSAX (provides trace of OSAX adapter)	"OSAX Adapter Trace" on page 62	"OSAX Adapter Trace" on page 99	-	"OSAX Adapter Trace" on page 151
PGMCHECK (provides trace of program checks)	"PGMCheck Trace (Program Check)" on page 63	"PGMCheck Trace" on page 100	"Program Check Trace" on page 126	"PGMCheck (Program Check) Trace" on page 152
PGMLOAD (provides trace of phase load requests, or actual load)	"PGMLOAD (Fetch/Load) Trace" on page 64	"Program Load Trace (Fetch/Load Trace)" on page 101	"Fetch/Load Trace" on page 125	"PGMLoad (Program Load) Trace" on page 152
SSCH (provides trace of start Subchannel instructions)	"SSCH Instruction Trace" on page 65	"Statement Examples" on page 103	"SSCH and I/O Interrupt Trace" on page 124	"Start Subchannel Instruction Trace" on page 154
STORAGE (provides trace of storage alterations)	"STORAGE Alteration Trace" on page 66	"Storage Alteration Trace" on page 103	"Storage Alteration Trace" on page 127	"STorage Alteration Trace" on page 154
SVC (provides trace of executed supervisor calls)	"SVC Trace (Supervisor Call)" on page 67	"Supervisor Call Trace" on page 105	"SVC Trace" on page 128	"SVC (Supervisor Call) Trace" on page 154

Table 3. Trace Type Summary (continued)

Trace Type	Des	Dir	Prc	Prp
VTAMBU (provides trace of usage of VTAM® buffers)	“VTAMBU Trace (VTAM Buffer)” on page 67	“VTAM BUffer Trace” on page 106	-	“VTAMBU (VTAM Buffer) Trace” on page 155
VTAMIO (provides trace of VTAM I/O operations)	“VTAMIO Trace” on page 68	“VTAMIO Trace” on page 107	-	“VTAMIO (VTAM I/O) Trace” on page 155
XPCC (provides trace of cross partition communication)	“XPCC Trace” on page 69	“XPCC Trace” on page 108	-	“XPCC (Partition Communication) Trace” on page 155

* See, however, “System Performance Degradation Caused by PER Traces” on page 49.

Trace Output

The trace output, an **event record,** is supplied for each occurrence of a traced event, according to your setup instructions.

You may request the event records to be written to a line printer, onto magnetic tape, or into a wraparound buffer. How to define the output device is described together with each type of initialization process.

Sample event records are shown for each trace type under “Summary of TRACE Types” on page 55.

Performance Considerations

System Performance Degradation

The tracing of events with SDAID may affect overall system performance. This may especially affect time dependent programs (such as programs doing input/output via telecommunication lines).

As long as SDAID processes a tracing event, all external and I/O interrupts are disabled and remain pending until trace data collection is complete. The supervisor may recognize an attention interrupt from the system console immediately or with a significant delay. Specification of an output tape (OUTDEV TAPE=cuu) reduces the possible time delays.

When you invoke SDAID in prompt mode, console input is blocked during the processing of each SDAID command.

System Performance Degradation Caused by PER Traces

The following SDAID traces use the program event recording (PER) feature:

- branch trace (BRANCH)
- instruction trace (INSTRUCTION)
- storage alter trace (STORAGE).

The PER feature allows the SDAID to limit the trace address range via the control registers 10 and 11. For this, the use of the address specification (ADDress= in direct input mode for example) helps to achieve better performance.

SDAID Space Requirements

Space Requirements during Initialization in the AR

The SDAID setup phases are loaded into the system GETVIS space. The SDAID setup phases require approximately 100K bytes (K equals 1,024) of virtual storage. When initialization is complete (the READY command is processed successfully), that GETVIS space is released.

Space Requirements during Initialization in a Partition

Beside the GETVIS space of 100K bytes the phase SDAID (called via EXEC SDAID) requires approximately 16K bytes of partition virtual storage. This is significantly less than the minimum VSE partition size. Therefore SDAID will run in any foreground or background partition.

Space Requirements for SDAID Execution

When the READY command is entered, SDAID allocates and fixes a certain number of pages in processor storage for SDAID execution. The amount of storage required for SDAID execution depends on the combination of trace operations that you request and on the size of the output buffer (specified in the OUTDEV command).

To execute SDAID, an area is allocated that is between:

- The supervisor area.
- The start-address of the SVA(24)

By default, the allocated SDAID area is 64K bytes. During IPL, you can increase the size of the SDAID area by using the SDSIZE parameter of the SYS command.

Here are some general guidelines for determining the required size of the SDAID area:

- For simple applications (where the OUTPUT parameter is not used), SDAID requires approximately 30K bytes.
- For more complex applications, SDAID requires approximately 60K bytes.

During execution, SDAID fixes a certain number of pages in processor storage. As a result, the number of page frames available to VSE for the execution of programs in virtual mode is reduced, which may affect overall system performance.

Space Requirements for the Buffer

The internal wrap-around buffer does not belong to the SDAID area. It is situated in the SVA (31) area, which avoids the problem that storage will be exhausted if a more complex SDAID trace and a BUFFER are specified. The buffer size is requested and prefixed as multiples of 4K bytes.

If you specify TAPE=cuu, the BUFFER parameter will not be used. SDAID will then allocate an internal default buffer of 8K bytes.

Space Requirements for SDAID Execution, Summary

```
Basic requirement for SDAID execution:          20K

Additional requirements
 Per specified trace:                           2K
 If BUFFER=nn is specified:                     2K + buffer size
 If OUTPUT is used and OUTDEV=Tape or Buffer    12K
 If OUTPUT is used and OUTDEV=Printer           20K
```

Number of Traces per Session

In prompt and direct input mode

The number of TRACE commands that you can submit per session depends on the types of the specified traces and the requested trace options. For each TRACE command, SDAID builds at least one TRACE command control block; for some it builds two such blocks as shown below. The program can build (and use) a maximum of ten TRACE command control blocks per session. The number of blocks per TRACE command is:

```
                PHase=phasename   PHase=phasename
Type of trace   not specified     specified
_____________   _______________   _______________
 PGMLOAD               2                  2
 VTAMIO                3                  3
 all others            1                  2
```

If the traces that you requested require more than ten trace control blocks, the program ignores the TRACE command that was submitted last and informs you about this action with a message.

Via procedures

Only one procedure statement is possible, but it can create one or more TRACE commands.

Chapter 8. SDAID General Description

This topic contains a general description of all SDAID initialization definitions, and output examples for all trace types.

The format of the commands and definition examples are shown under the descriptions for the various initialization methods in Chapter 7, "SDAID Overview," on page 45, Chapter 8, "SDAID General Description," on page 53, and Chapter 9, "Initialize an SDAID Trace in Direct Input Mode," on page 83.

Defining the Output Device

The following output destinations can be defined when you initialize an SDAID trace:

- Printer
- Tape
- Wraparound buffer
- Wraparound buffer and printer
- Wraparound buffer and tape

You define the output destination for the event records in prompt and direct input mode via the 'OUTDEV' statement. If you use a procedure to initialize a trace, the 'BUFFER=', 'TAPE=', or 'PRINTER=' specifications are used to accomplish an OUTDEV definition.

Printer Defined as Output Destination

If a line printer is defined for the output, SDAID writes the event records on the printer at the time the particular events occur.

If any program in the system writes output directly to the printer, this output will be mixed with the SDAID output. You can avoid this by

- Collecting trace output on tape and printing it afterwards via DOSVSDMP, or
- Stopping the VSE/POWER controlled printer during tracing.

Tape Defined as Output Destination

If you define a magnetic tape as output device, the SDAID program moves the trace entries into an internal buffer. SDAID writes event records to the available tape volume whenever this buffer becomes full, or a 'STOPSD' or 'ENDSD' command is processed.

Every STOPSD or ENDSD command writes a tapemark on the tape if there was any trace event. However, if there was no trace event since the last STARTSD command, the tape remains unchanged.

SDAID writes the buffers successively to the tape, and one trace entry can extend over several buffers. For this reason, if you change reels after end-of-volume, the second tape may start with an incomplete trace entry.

Buffer Defined as Output Destination

A buffer in storage may be defined to store the trace event records. During tracing, SDAID stores one event record after the other. When the buffer becomes full, SDAID wraps around and continues to write event records at the beginning of the buffer overwriting previously stored records. A buffer used in this way is called a *wraparound buffer*.

A wraparound buffer is not automatically printed when it is full. SDAID writes the buffer contents on a printer or on a tape if you request this action explicitly:

- The contents of the buffer will be written to the output device if a buffer trace is specified (TRACE BUFFER). The buffer trace causes the buffer to be written whenever it is full.
- The contents of the buffer will be written to the output device if you specify OUTPut=BUffer on a TRACE statement. The specification OUTPut=BUffer causes the buffer to be written whenever the event specified in the TRACE statement occurs.
- The contents of the buffer will be written to the output device if you use the BUFFER and BUFFOUT keyword in an SDAID procedure.

The information contained in the buffer can also be retrieved with the attention routine DUMP command (see Figure 19 on page 54 and "Printing an SDAID or DUMP Command Produced Tape" on page 31).

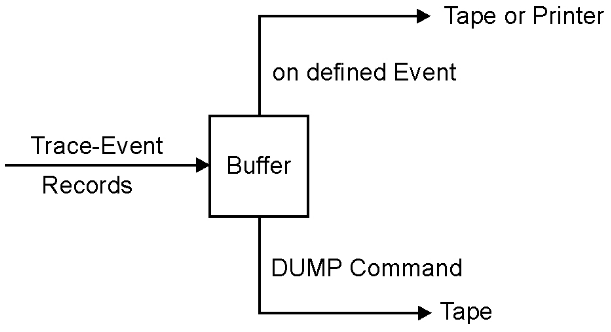


Figure 19. Overview, Tracing Events into a Buffer

You **define the size** of the buffer in number of blocks of 4K bytes.

The possible buffer size which you can define depends on the device type of the buffer output device.

If you define a tape device, the possible size may vary from 4K to 32K bytes.

You can request a buffer size from 4K to 256K bytes if a printer is defined as output device, or if no output device is defined.

Steps to Define a Wraparound Buffer

Assume, for example, that you want to:

- Collect the event records of an instruction trace in a 6K bytes buffer as long as no program check occurs;
- Write the contents of the buffer to a tape when a program check interruption occurs.

Perform the following steps:

1. Define the buffer and the tape device with the OUTDEV command (in direct input or prompt mode) or with the procedure statement in the form: 'BUFFER=6 TAPE=280';
2. Define the instruction trace;
3. Define the PGMCHECK trace with OUTPUT=BUFFER (in direct input or prompt mode), or define BUFFOUT=PGMC in the procedure statement.

For the formats of the output definitions, see the following sections according to the input mode used:

- "Defining the Output Device in Direct Input Mode" on page 86;
- "BUFFER=, PRINTER=, TAPE=Keyword Operands" on page 130;
- "Output Device Definition in Prompt Mode: OUTDEV Command" on page 144.

Exceptional Conditions on the Output Device

You define the SDAID output device via the OUTDEV statement. It is required that the specified printer or tape device is ready when you activate tracing via the STARTSD statement.

If an exceptional condition occurs on the output device, SDAID performs the following actions:

- Unrecoverable I/O error on tape or printer

  SDAID resets the control registers to the previous state and stops trace data collection. The VSE system continues normal operation without tracing. On the next STOPSD statement SDAID presents an error message with an error code as shown in the publication z/VSE Messages and Codes Volume 1. A final ENDSD statement releases all resources allocated to SDAID.
- End-of-volume condition on tape

  SDAID writes two trailing tape marks to close the tape file, performs a 'rewind-unload' operation, and enters a soft wait state with the value X'00EEEEEE' in the address part of the wait PSW. The operator should now perform the following actions:

  – Mount a new tape reel
  – Make the tape device ready
  – Press the external interrupt key

  SDAID will continue tracing onto the new tape reel.
- Intervention required on printer device (printer out of paper or stopped manually)

  SDAID waits about two minutes to allow for paper refilling or other actions on the printer device. After this time has elapsed, SDAID enters a soft wait state with the value X'00EEEEEE' in the address part of the wait PSW. The operator should now perform the following actions:

  – Make the printer device ready
  – Press the external interrupt key

  SDAID continues tracing on the printer device.
- Intervention required on tape device

  SDAID enters a soft wait state with the value X'00EEEEEE' in the address part of the wait PSW. The operator should now perform the following actions:

  – Make the tape device ready
  – Press the external interrupt key

  SDAID continues tracing to the tape device.

Note: If the system is in the soft wait state with X'00EEEEEE' in the address part of the PSW and the operator presses the interrupt key without making the SDAID output device ready, SDAID stops trace data collection. (SDAID reacts in the same way as for unrecoverable I/O errors).

Summary of TRACE Types

SDAID offers you various trace types so that you get the most suitable information for solving the problem in hand.

The following sections describe all SDAID trace types with their SDAID default values and shows trace event record examples. The shown output may be written to a buffer or to a tape or printer device, according to your output device specification.

Table 4 on page 55 lists the commands which produce the different trace types, and summarizes the event traced in response to each command. The references in this table help you to find the description and an output example of each trace type.

Table 4. Trace Type Summary

Trace Type	Provides a Trace of:	See:
BRANCH	Successfully executed branch instructions	"BRANCH Trace" on page 56
BUFFER	The trace buffer when it is full	"BUFFER Trace" on page 57

Table 4. Trace Type Summary (continued)		
Trace Type	**Provides a Trace of:**	**See:**
CANCEL	Program (main task) cancel or EOJ	"CANCEL Trace" on page 57
EXTERNAL	External interrupts	"EXTERNAL Trace" on page 57
EXTERNAL	External interrupts	"EXTERNAL Trace" on page 57
GETVIS	GETVIS / FREEVIS requests	"GETVIS Trace" on page 92
IO	I/O interrupts	"IO Trace (I/O Interrupt)" on page 59
LOCK	LOCK / UNLOCK requests	"LOCK Trace" on page 96
MONITORCALL	MC instructions	"MONITORCALL Trace" on page 61
OSAX	OSAX adapter	"OSAX Adapter Trace" on page 62
PGMCHECK	Program checks	"PGMCheck Trace (Program Check)" on page 63
PGMLOAD	Phase load requests, or actual load	"PGMLOAD (Fetch/Load) Trace" on page 64
SSCH	Start Subchannel instructions	"SSCH Instruction Trace" on page 65
STORAGE	Storage alterations	"STORAGE Alteration Trace" on page 66
SVC	Executed supervisor calls	"SVC Trace (Supervisor Call)" on page 67
VTAMBU	Usage of VTAM buffers	"VTAMBU Trace (VTAM Buffer)" on page 67
VTAMIO	VTAM I/O operations	"VTAMIO Trace" on page 68
XPCC	Cross partition communication	"XPCC Trace" on page 108

BRANCH Trace

A branch-instruction trace provides an event record for every branch taken, if the branch target address falls into the defined area.

An example of the output is shown in Figure 20 on page 56.

BRANCH Trace Output Example

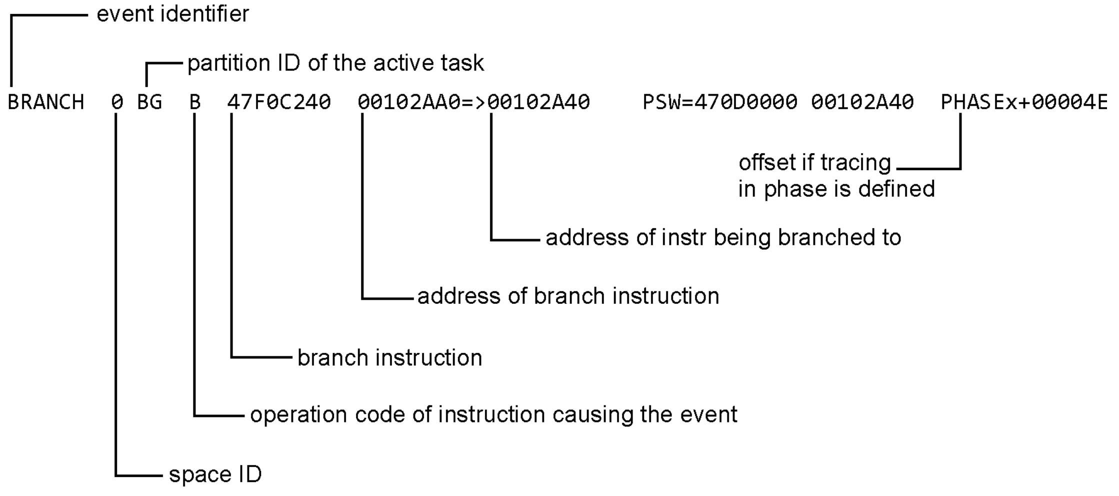

event identifier
partition ID of the active task
BRANCH 0 BG B 47F0C240 00102AA0=>00102A40   PSW=470D0000 00102A40 PHASEx+00004E
offset if tracing in phase is defined
address of instr being branched to
address of branch instruction
branch instruction
operation code of instruction causing the event
space ID

Figure 20. BRANCH Trace Event Record

BUFFER Trace

The buffer trace dumps the contents of the SDAID wraparound buffer to the output device (printer or tape) when the buffer is full.

The buffer trace can be used only if you have also ‘Printer’ or ‘Tape’ specified in the OUTDEV command or in a procedure.

The buffer trace output is the collection of all trace records contained in the buffer when a buffer overflow occurs. These trace records are written sequentially.

CANCEL Trace

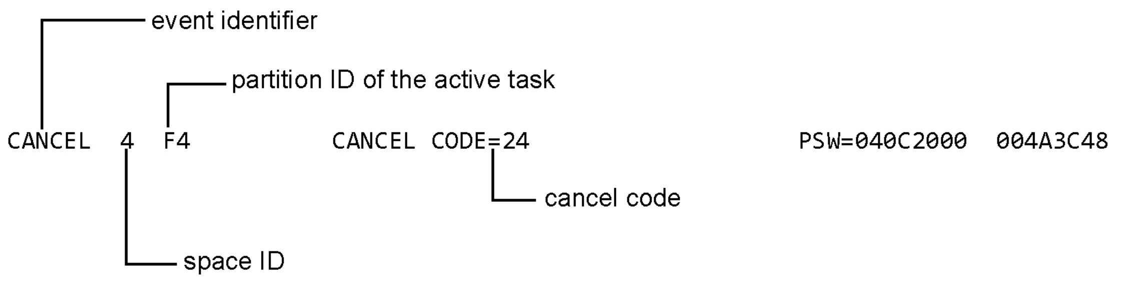
This trace provides an event record when the main (or only) task of the traced partition is canceled or reaches EOJ.

You may use this trace type combined with additional output definitions to get more reasonable information at the time of a cancel or EOJ condition.

For example, use the cancel trace type to get the buffer or areas of interest together with the cancel event record recorded.

An example of the cancel trace output is shown in Figure 21 on page 57.

CANCEL Trace Output Example



```
CANCEL  4  F4         CANCEL  CODE=24                   PSW=040C2000  004A3C48
```

Figure 21. CANCEL Trace Event Record

EXTERNAL Trace

The external trace provides information concerning the occurrences of external interrupts such as pressing the external interrupt key. You may define one to eight of the following external interrupt types:

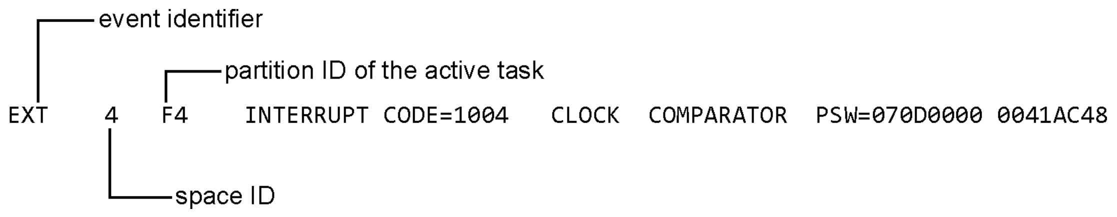
```
0040   Interrupt key
1003   TOD-clock sync check
1004   Clock comparator
1005   CPU timer
1200   Malfunction alert
1201   Emergency signal
1202   External call
2401   Service signal
2402   Logical device           * z/VM CP
2603   PFAULT handshaking       * z/VM CP
4000   IUCV, APPC               * z/VM CP
4001   VMCF                     * z/VM CP
```

SDAID Default Value

If you do not define the type of interrupt, all external interrupts are traced.

The format of a printed external-interrupt trace event record is shown in Figure 22 on page 58.

EXTERNAL Interrupt Trace Output Example



```
EXT     4   F4    INTERRUPT CODE=1004    CLOCK   COMPARATOR   PSW=070D0000 0041AC48
```

Figure 22. EXTERNAL Interrupt Trace Event Record

GETVIS / FREEVIS Trace

A GETVIS / FREEVIS trace provides information about requests made to obtain or release virtual storage. These requests can be made using:

- SVC 3D
- SVC 3E
- An internal GETVIS call via SGETVIS and SFREEVIS macros.

The simple trace of the SVC's 3D and 3E only show the existence of SVCs at the point of invocation. However, the GETVIS / FREEVIS trace records the results of a virtual-storage request *after* it has been evaluated by the z/VSE GETVIS / FREEVIS routines.

You can limit the tracing of your GETVIS / FREEVIS requests to:

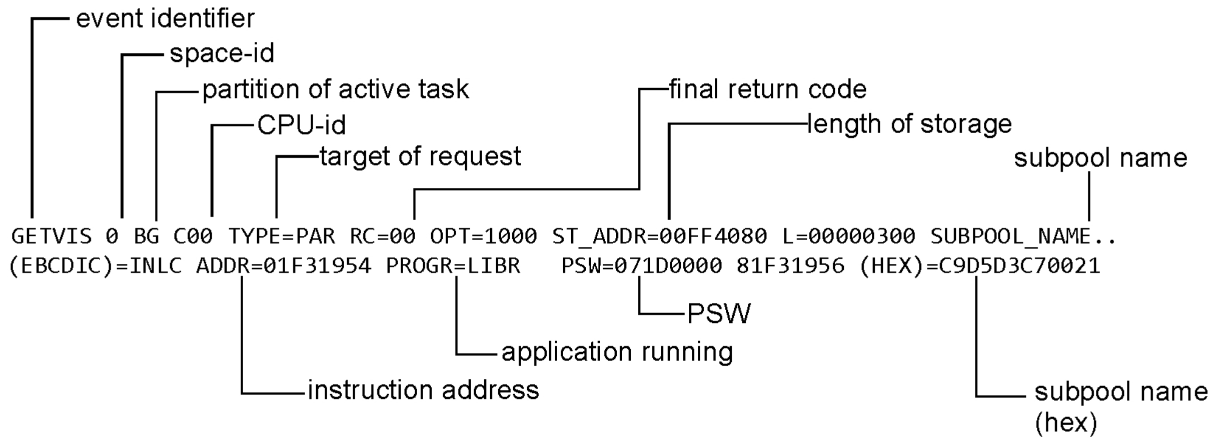
- A specific partition.
- The supervisor.
- A specific subpool name.
- A GETVIS location (24-bit or 31-bit area).

SDAID Default Value

If you do not define a partition or the supervisor, all tasks of the system are traced. All GETVIS / FREEVIS requests are traced if you do not specify any of the GETVIS trace definitions.

The format of a GETVIS / FREEVIS event record is shown in Figure 23 on page 58.

GETVIS / FREEVIS Trace Output Example



```
GETVIS 0 BG C00 TYPE=PAR RC=00 OPT=1000 ST_ADDR=00FF4080 L=00000300 SUBPOOL_NAME..
(EBCDIC)=INLC ADDR=01F31954 PROGR=LIBR   PSW=071D0000 81F31956 (HEX)=C9D5D3C70021
```

```
 FREEVS 0 BG C00 TYPE=PAR RC=00 OPT=1000 ST_ADDR=00FF8000 L=00001000 SUBPOOL_NAME..
(EBCDIC)=INLC ADDR=000C997E PROGR=LIBR   PSW=071D0000 800C9980 (HEX)=C9D5D3C70021
```

Figure 23. GETVIS / FREEVIS Trace Record

You can also display additional lines, as shown in Figure 24 on page 59.

```
FREEVS 0 BG C00 TYPE=   RC=00 OPT=220C --ENTIRE STORAGE RELEASED--
ADDR=00609CB2 --JCL PHASE ACTIVE-- PSW=070D0000 00609CB4

FREEVS 0 BG C00 TYPE=PAR RC=00 OPT=2002 --ENTIRE SUBPOOL RELEASED-- SUBPOOL_NAME..
(EBCDIC)=SP01 ADDR=006000A2 PROGR=GETV  PSW=070D0000 006000A4  (HEX) =E2D7F0F14040
```

Figure 24. Additional Fields Displayed By GETVIS / FREEVIS Trace

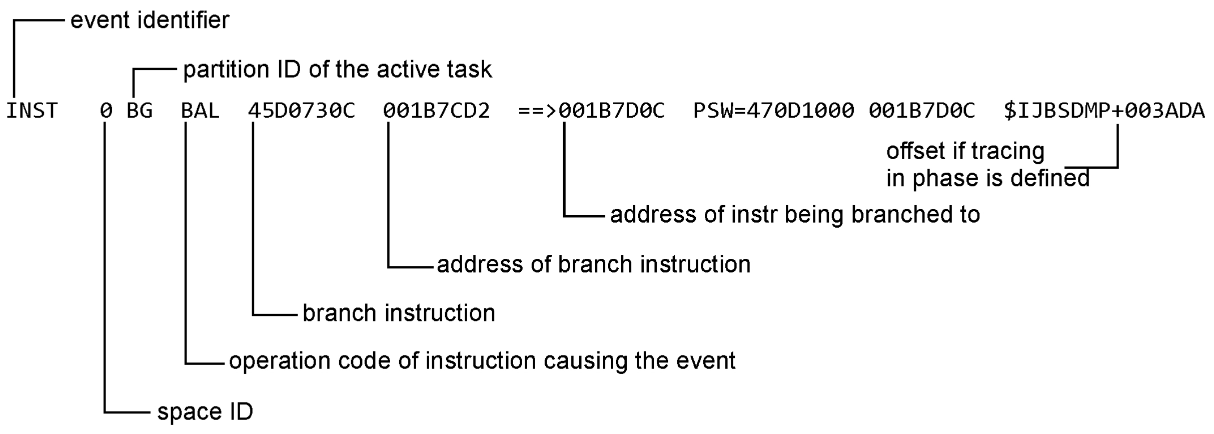
INSTRUCTION Trace

An instruction trace provides information for instructions executed in the area defined for the trace. You may select certain types of instructions to be traced by defining the instruction operation codes or all types by defining an asterisk (*). Moreover, the trace can be defined to record all branch instructions, without regard to whether the branch is taken or not.

Note the difference from the branch trace, which only records the branches actually taken.

The format of a printed event record for an instruction trace is shown in Figure 25 on page 59.

INSTRUCTION Trace Output Example



```
INST    0 BG  BAL  45D0730C  001B7CD2  ==>001B7D0C  PSW=470D1000 001B7D0C  $IJBSDMP+003ADA
```

event identifier
partition ID of the active task
offset if tracing in phase is defined
address of instr being branched to
address of branch instruction
branch instruction
operation code of instruction causing the event
space ID

```
INST    0 BG  TM   91801002  001B7D0E               PSW=470D3000 001B7D12  $IJBSDMP+003B16
```

Figure 25. INSTRUCTION Execution Event Record

IO Trace (I/O Interrupt)

The IO trace collects information about I/O interrupts.

You may limit the I/O operations to be traced to a partition or to the supervisor. Another limitation is to define a particular unit, control unit, or channel to be traced.

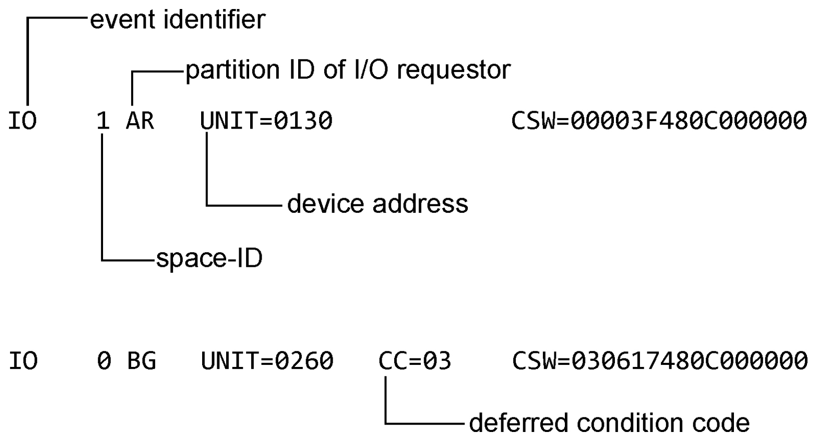
SDAID Default Value

If you do not define a partition or the supervisor, all tasks of the system are traced. All I/O devices are traced if you do not specify any of the I/O definitions.

The format of an I/O-interrupt event record is shown in Figure 26 on page 60.

Note: Due to the fact that no tables are available in the supervisor for TP status modifier commands, the output of TP channel programs (e.g. VTAM) may be incomplete.

I/O Interrupt Trace Output Example



```
IO    1 AR   UNIT=0130            CSW=00003F480C000000

IO    0 BG   UNIT=0260   CC=03    CSW=030617480C000000
```

Figure 26. I/O-Interrupt Trace Event Record

LOCK / UNLOCK Trace

A LOCK / UNLOCK trace provides information about requests made to lock or unlock a resource.

You can limit the tracing of your LOCK / UNLOCK requests to:

- A specific partition.
- The supervisor.
- A specific resource name.
- A lock type.
- The scope of the lock request.
- A volume ID.
- A dedicated return code.

SDAID Default Value

If you do not define a partition or the supervisor, all tasks of the system are traced. All LOCK / UNLOCK requests are traced if you do not specify any of the LOCK trace definitions.

The format of a LOCK / UNLOCK event record is shown in Figure 27 on page 61.

LOCK / UNLOCK Trace Output Example

The three trace output lines below are listed separately, to enable explanations to be given.

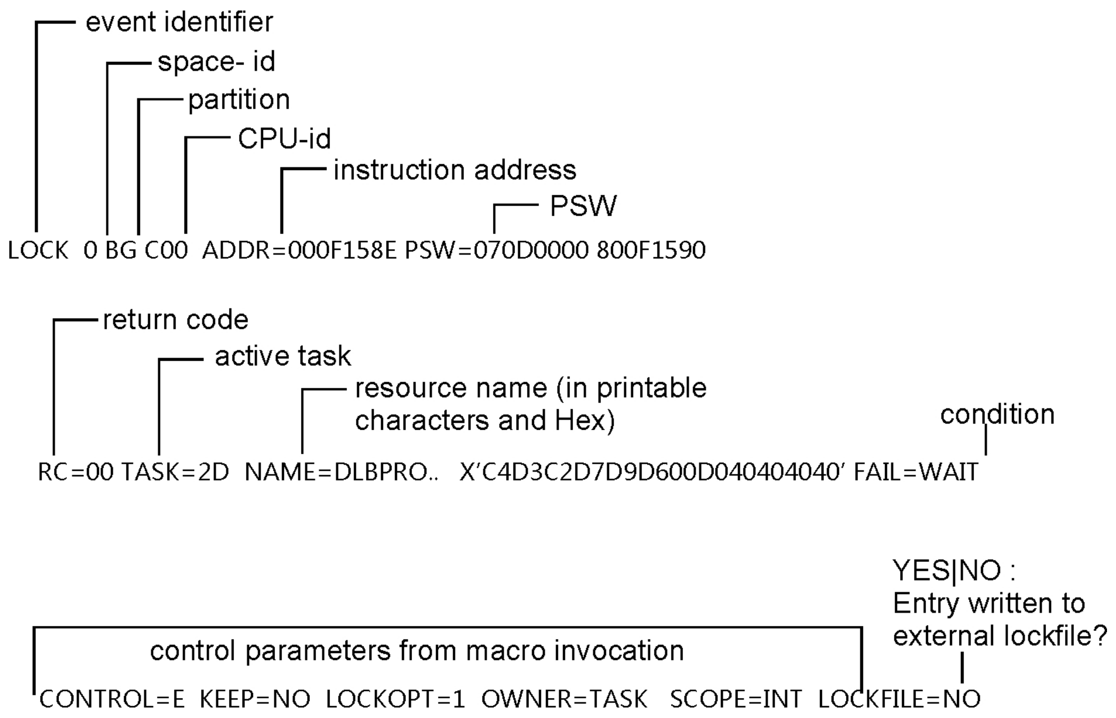

event identifier
space- id
partition
CPU-id
instruction address
PSW
LOCK 0 BG C00 ADDR=000F158E PSW=070D0000 800F1590
return code
active task
resource name (in printable characters and Hex)
condition
RC=00 TASK=2D NAME=DLBPRO.. X'C4D3C2D7D9D600D040404040' FAIL=WAIT
YES|NO : Entry written to external lockfile?
control parameters from macro invocation
CONTROL=E KEEP=NO LOCKOPT=1 OWNER=TASK SCOPE=INT LOCKFILE=NO

Figure 27. LOCK Trace Record

You can also display a line containing the contents of the LOCKTABLE entry, as shown in Figure 28 on page 61.

```
---LTE---
000619B0 00000000 00000000 00000000 00000000 00000000 00000000 000619D0 00000000
```

Figure 28. Contents of LOCKTABLE

MONITORCALL Trace

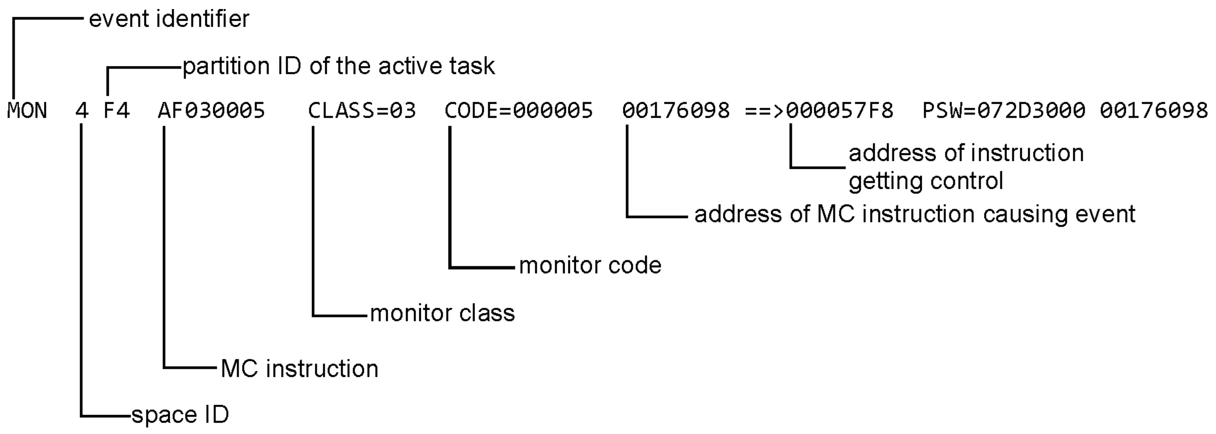
The monitor call trace provides information about monitor call instruction executions.

You may define all (defined via an asterisk (*)) or up to eight mc (monitor classes) in hexadecimal notation of the MC instructions to be traced. An event record is provided when an executed MC instruction has a monitor class which matches any of the specified classes.

You may specify any valid monitor class; however, SDAID ignores a specification of class 2, because class 2 is used by SDAID to control tracing.

The format of an MC instruction trace event record is shown in Figure 29 on page 62.

MONITORCALL Trace Output Example



```
MON   4 F4   AF030005    CLASS=03   CODE=000005   00176098 ==>000057F8   PSW=072D3000 00176098
```

event identifier
partition ID of the active task
address of instruction getting control
address of MC instruction causing event
monitor code
monitor class
MC instruction
space ID

Figure 29. MONITORCALL Trace Event Record

OSAX Adapter Trace

The OSAX adapter trace provides useful information about the status of an OSAX adapter that is used in your z/VSE system.

Using the OSAX adapter trace, you can either:

- specify an address of *one* data path only, or
- trace *all* data path addresses (the default).

The format of an OSAX adapter trace record is shown in Figure 30 on page 63.

OSAX Adapter Trace Output Example

Figure 30 on page 63 shows an example of the trace output that is generated by the following statements:

```
// EXEC SDAID
 OUTDEV P=01E
 TRACE OSAX EXT=Y
/&
```

```
START OF SDAID TRACE
OSAX   8  F8  C00  DATAPATH=0E02  FUNCTION=SEND     SUB_FUNCTION=SEND_NOLST
---XMIT_PARAMETER_LIST---
005D283C 034A3F80 0000005C 09985001 035CB000
---IP_HEADER-------------
034A3F80 4500005C 00060000 FF010454 0998520C 0998520B 080013D6 00210005 00010203
OSAX   8  F8  C00  DATAPATH=0E02  FUNCTION=SEND     SUB_FUNCTION=SEND_FHDR
---OSA_HEADER------------
03598000 01060000 00000000 005C0000 00000000 00000000 00000000 00000000 09985001
OSAX   8  F8  C00  DATAPATH=0E02  FUNCTION=SEND     SUB_FUNCTION=SEND_SPNL
---RETURN CODE-----------
035CE248 00000004
---REASON CODE-----------
035CE278 0000
OSAX   8  F8  C00  DATAPATH=0E02  FUNCTION=SEND     SUB_FUNCTION=SEND_NOLST
---XMIT_PARAMETER_LIST---
005D283C 00000000 00000000 09985001 035CB000
OSAX   8  F8  C00  DATAPATH=0E02  FUNCTION=SEND     SUB_FUNCTION=SEND_QDIO
---SLSB------------------
035C9A00 A1A1A1A1 A162A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0
035C9A20 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0
035C9A40 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0
035C9A60 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0 A0A0A0A0
---RETURN CODE-----------
035D1CD4 00000000
---REASON CODE-----------
035CE278 0000
---SBAL------------------
035CA8F0 00000000 00001000 00000000 0127B000
OSAX   8  F8  C00  DATAPATH=0E02  FUNCTION=SEND     SUB_FUNCTION=SEND_SPNL
---RETURN CODE-----------
035CE248 00000000
OSAX   8  F8  C00  DATAPATH=0E02  FUNCTION=SEND     SUB_FUNCTION=SEND_NOLST
   :
END OF SDAID OUTPUT
```

Figure 30. Example of OSAX Adapter Trace Event Record

Please note that the output from an OSAX adapter trace is complex and normally only suitable for use by IBM personnel. Typically, an OSAX adapter trace would be requested by IBM support personnel after a related problem has been reported to them.

PGMCheck Trace (Program Check)

The program check trace provides information on the occurrence of program check interrupts.

You may limit the trace operation by defining certain program interruption codes. Up to 16 program interruption codes of a value lower than X'40' in hexadecimal notation may be defined.

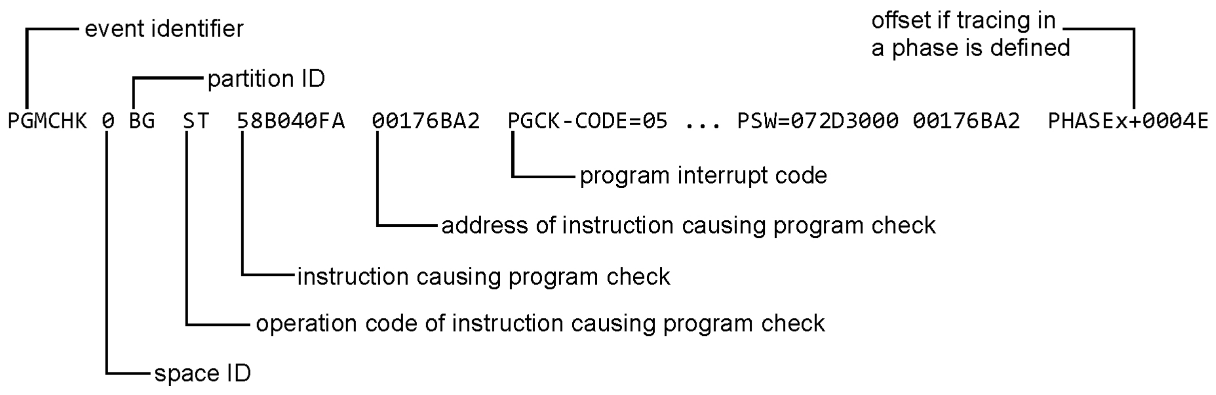
SDAID writes an event record only if the interrupt code returned by the system matches one of the specified interrupt codes.

If you do not want to limit the trace recording to a specific interrupt code, define an asterisk (*) to trace all program checks - except those page or segment translation exceptions which are caused by the temporary absence of a storage page. The specification PGMC=(10 11) traces all page or segment translation exceptions.

For a discussion of program interrupt codes, refer to the applicable *Principles of Operation* manual.

The format of a program-check event record is shown in Figure 31 on page 64.

PGMCHECK Trace Output Example



```
PGMCHK 0 BG  ST  58B040FA  00176BA2  PGCK-CODE=05 ... PSW=072D3000 00176BA2  PHASEx+0004E
```

event identifier
partition ID
offset if tracing in a phase is defined
program interrupt code
address of instruction causing program check
instruction causing program check
operation code of instruction causing program check
space ID

Figure 31. Program Check Trace Event Record

The program check trace also displays some additional fields, as shown in the following examples:

```
PGMCHK 0  BG  ST  58B040FA  00176BA2   PGCK-CODE=10  TEID=00581800  EAID=01  PSW=072D3000 00176BA2
              RADD=xxxxxxxx    STE=xxxxxxxx
PGMCHK 0  BG  ST  58B040FA   00176BA2  PGCK-CODE=10  TEID=00581800  EAID=01  PSW=072D3000 00176BA2
              RADD=xxxxxxxx
PGMCHK 0  BG  ST  58B040FA  00176BA2   PGCK-CODE=11  TEID=00581800  EAID=01  PSW=072D3000 00176BA2
              RADD=xxxxxxxx    PTE=xxxxxxxx
PGMCHK 0  BG  ST  58B040FA  00176BA2   PGCK-CODE=2A                 EAID=01  PSW=072D3000 00176BA2
```

TEID=xxxxxxxx specifies the Translation Exception Identification. SDAID retrieves this information from low core location 144-147. **EAID**=xx shows the Exception Access Identification. SDAID retrieves this information from low core location 160. SDAID displays the fields TEID and/or EAID if the processor updates the low core locations 144-147 and/or 160 at interrupt time.

If the programming exception is a page or a segment translation exception, SDAID displays the field RADD and (if applicable) one of the fields PTE or STE. If the page or segment table portion of the virtual address points inside the page or segment table, **RADD**=xxxxxxxx shows the real address of the invalid page or segment table entry. The field **PTE**=xxxxxxxx shows the invalid page table entry. The field **STE**=xxxxxxxx shows the invalid segment table entry. If the page or segment table portion of the virtual address points outside the page or segment table, RADD shows the real address of the entry that would have been fetched if the length violation had not occurred. In this case SDAID displays no PTE or STE field.

PGMLOAD (Fetch/Load) Trace

The program load trace provides information about program load events.

Such a program load event can be one of the following:

- Phase fetch/load request
- Completion of fetch/load operation

You may limit the trace recording if you define that only the fetch/load request (REQ) or only the actual fetch/load completion (HDL) is to be traced.

You can limit the trace data collection to the load events of a certain phase.

You can limit the trace data collection to the load events occurring in an address range. This means that SDAID records

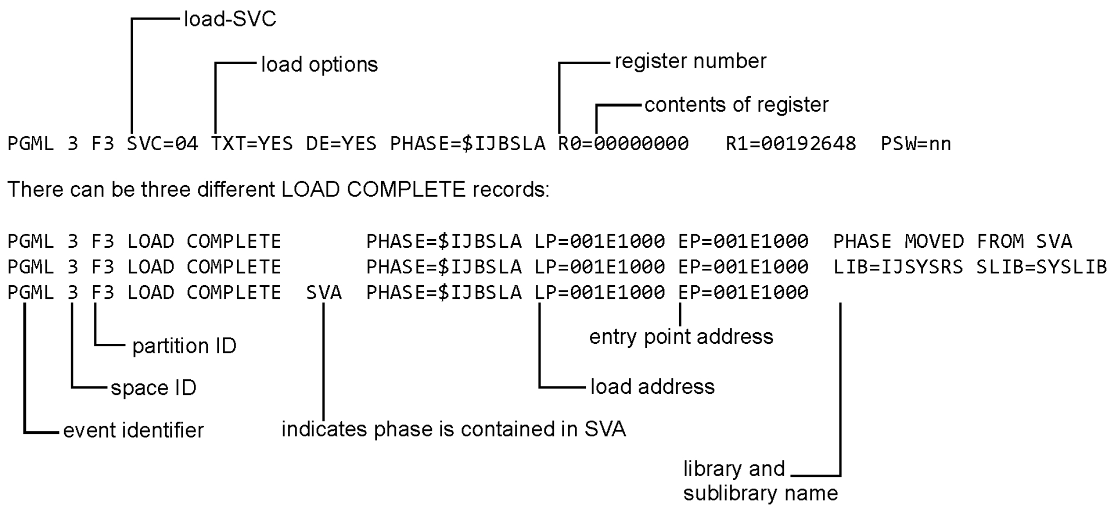
- The trace load SVCs issued within the specified address range,
- The trace load completion events if the phase is loaded into the specified address range.

SDAID Default Values

If you do not define the kind of the fetch/load request, both the request and its handling is traced. All phases are traced if you do not define a specific phase.

The format of a program load event record is shown in Figure 32 on page 65.

PGMLOAD Trace Output Example



```
PGML 3 F3 SVC=04 TXT=YES DE=YES PHASE=$IJBSLA R0=00000000   R1=00192648  PSW=nn
```

There can be three different LOAD COMPLETE records:

```
PGML 3 F3 LOAD COMPLETE       PHASE=$IJBSLA LP=001E1000 EP=001E1000  PHASE MOVED FROM SVA
PGML 3 F3 LOAD COMPLETE       PHASE=$IJBSLA LP=001E1000 EP=001E1000  LIB=IJSYSRS SLIB=SYSLIB
PGML 3 F3 LOAD COMPLETE  SVA  PHASE=$IJBSLA LP=001E1000 EP=001E1000
```

Figure 32. PGMLOAD Trace Event Records

SSCH Instruction Trace

The SSCH instruction trace provides a trace event record for every executed start subchannel (SSCH) instruction.

You may limit the trace operation by defining selected tasks or by defining a unit, a control unit, or a channel address. A partition or the supervisor may be defined as the traced tasks.

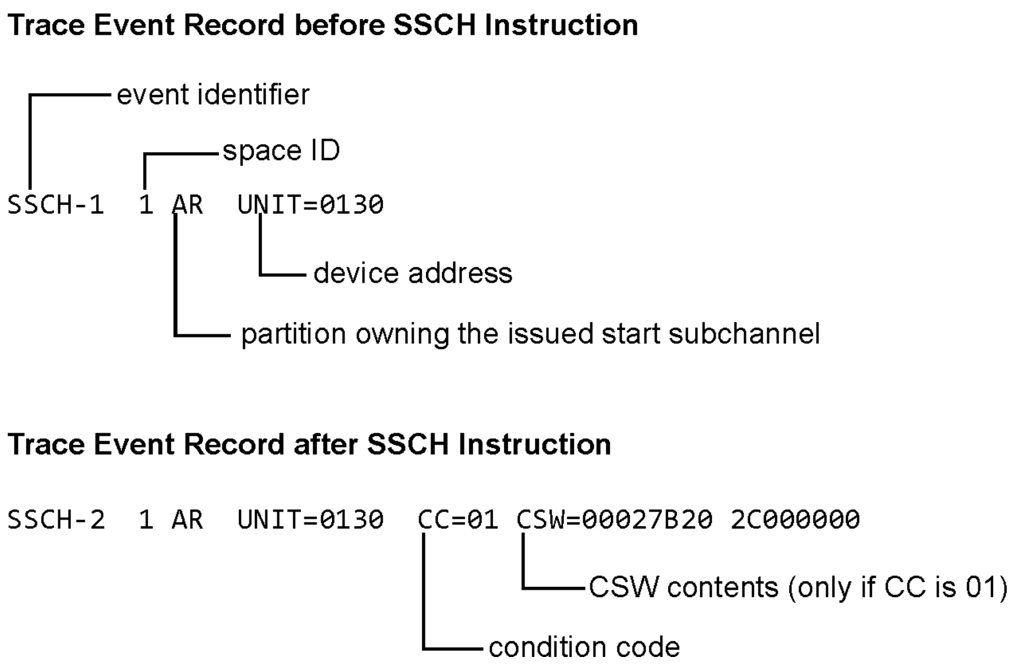
The SSCH trace may print two types of SSCH records. The SSCH-1 record shows the state before the SSCH has been performed. SDAID always displays a SSCH-1 record. The SSCH-2 record shows the condition code after the SSCH instruction. SDAID displays a SSCH-2 record if the condition code is different from zero; if the condition code is equal to zero, the SSCH-2 record is suppressed.

SDAID Default Value

If you do not define a partition or the supervisor, all tasks of the system are traced. All I/O devices are traced if you do not specify any of the I/O definitions.

The format of a printed standard SSCH instruction event record is shown in Figure 33 on page 66.

Note: Due to the fact that no tables are available in the supervisor for telecommunications status modifier commands, the output of telecommunications channel programs may be incomplete.


Trace Event Record before SSCH Instruction
event identifier
space ID
device address
partition owning the issued start subchannel
Trace Event Record after SSCH Instruction
CSW contents (only if CC is 01)
condition code

```
SSCH-1   1 AR  UNIT=0130
```

```
SSCH-2   1 AR  UNIT=0130  CC=01 CSW=00027B20 2C000000
```

Figure 33. SSCH (Start Subchannel) Trace Event Record

STORAGE Alteration Trace

The storage alteration trace writes an event record whenever a program alters the contents of a defined storage area. Storage alterations caused by I/O operations are not recorded.

With the AREA or JOBNAME specification you define the tasks you want to watch (the source space or partition). AREA=ALL causes all tasks of the system to be watched. With the STAREA, STJNAM, and STDSPN operands you define the altered storage area (the target area). If the source space (or source partition) is the same as the target space (or target partition), you may leave out the parameters STAREA, STJNAM and STJNUM. Via the ADDRESS definition you specify the storage range where the alteration occurs.

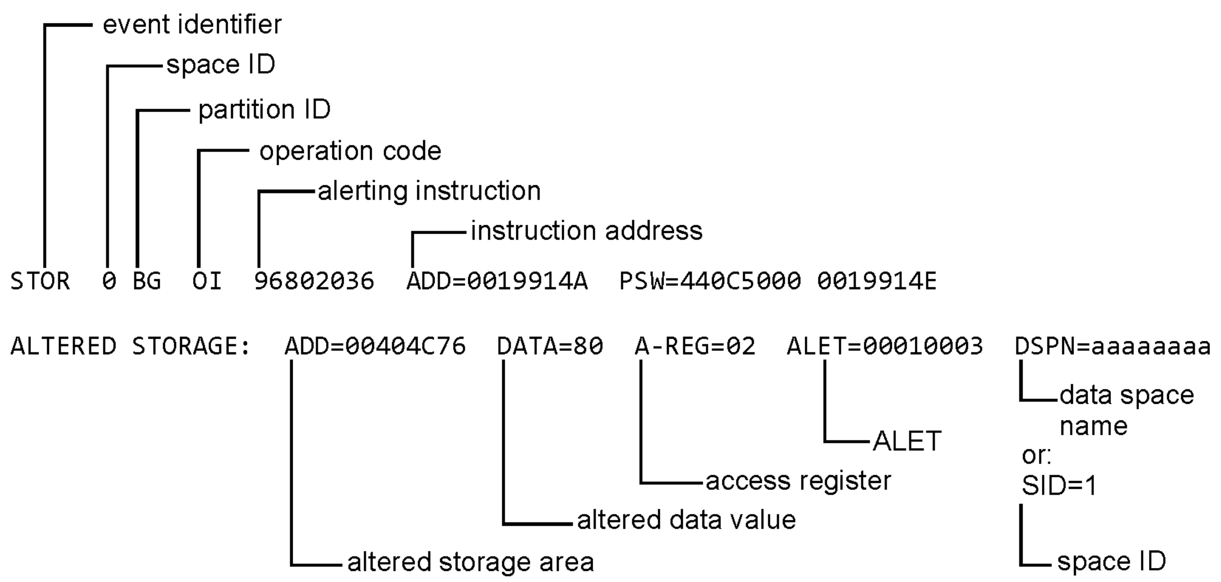
You may limit the trace operation by defining a certain storage *pattern*. If you define such a pattern, a trace event record is written only when a storage area is set to the specified value. Specify the pattern in hexadecimal notation. The pattern can be up to four bytes long.

If you specify an odd number of digits, a zero is inserted to the left of the first specified hexadecimal digit.

SDAID Default Value

You may omit the pattern definition. This causes each alteration of the defined storage interval to be traced.

An example of a storage alteration trace event record is shown in Figure 34 on page 67.



```
STOR   0 BG   OI   96802036   ADD=0019914A   PSW=440C5000 0019914E

ALTERED STORAGE:   ADD=00404C76   DATA=80   A-REG=02   ALET=00010003   DSPN=aaaaaaaa
```

Figure 34. Storage-Alter Trace Event Record

The first line of the storage-alteration trace record displays information about the altering program. The record shows the space-id, the partition-id, the altering instruction and its storage address, and the PSW. The second line gives information about the altered storage area. The record shows the address of the altered storage area (ADD=) and the altered data value (DATA=).

If the traced instruction has accessed data within a different address space, the record shows the access register number (A-REG=), the access register contents (ALET=), and the space identifier (SID=).

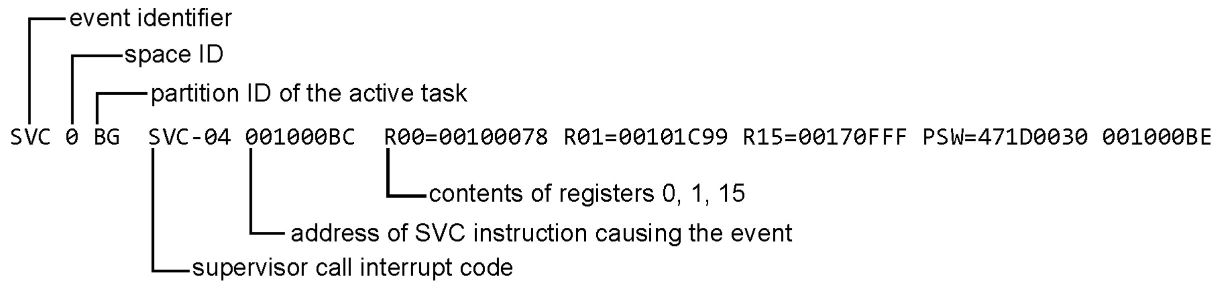
If the traced instruction has accessed a storage location within a data space, the record shows the access register number (A-REG=), the access register contents (ALET=), and the name of the target data space name (DSPN=).

SVC Trace (Supervisor Call)

The supervisor call trace provides information about one, several, or all SVC instructions executed in the defined area.

You have to define at least one or may define up to 16 different SVC codes or an asterisk (*) specifying that all supervisor call instructions are to be traced. Specify the SVC code in hexadecimal notation.

A typical SVC trace event record is shown in Figure 35 on page 67.



```
SVC 0 BG  SVC-04 001000BC  R00=00100078 R01=00101C99 R15=00170FFF PSW=471D0030 001000BE
```

event identifier
space ID
partition ID of the active task
contents of registers 0, 1, 15
address of SVC instruction causing the event
supervisor call interrupt code

Figure 35. SVC Trace Event Record

VTAMBU Trace (VTAM Buffer)

The VTAM buffer trace provides an event record each time VTAM uses one of the buffers in its buffer pool.

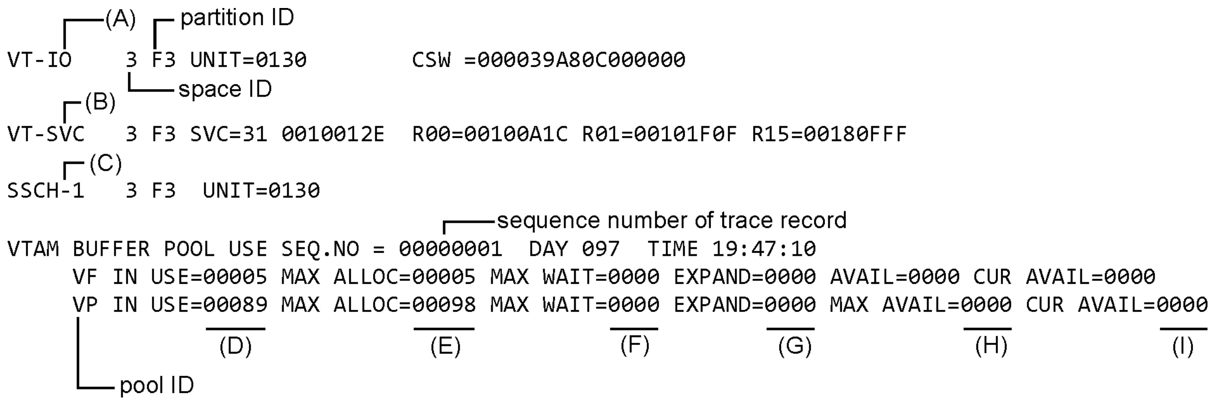
The format of a printed standard VTAMBU event record is shown in Figure 36 on page 68.

Note: For the VTAMBU trace you must condition VTAM either by entering a VTAM MODIFY command or by specifying the TRACE operation when you start VTAM. At the end of an SDAID session, end VTAM conditioning with the appropriate MODIFY command:

Start: F NET,TRACE,TYPE=SMS,ID=VTAMBUF

Stop: F NET,NOTRACE,TYPE=SMS,ID=VTAMBUF

For information on conditioning VTAM, see the *VTAM Diagnosis* publication.



```
VT-IO     3 F3 UNIT=0130        CSW =000039A80C000000

VT-SVC    3 F3 SVC=31 0010012E  R00=00100A1C R01=00101F0F R15=00180FFF

SSCH-1    3 F3  UNIT=0130

VTAM BUFFER POOL USE SEQ.NO = 00000001  DAY 097  TIME 19:47:10
     VF IN USE=00005 MAX ALLOC=00005 MAX WAIT=0000 EXPAND=0000 AVAIL=0000 CUR AVAIL=0000
     VP IN USE=00089 MAX ALLOC=00098 MAX WAIT=0000 EXPAND=0000 MAX AVAIL=0000 CUR AVAIL=0000
```

(A)
: I/O Interrupt Trace (for description see Figure 26 on page 60).

(B)
: SVC Trace (for description see Figure 35 on page 67).

(C)
: SSCH Trace (for description see Figure 33 on page 66).

(D)
: Number of buffers (pages for VF and VP) in use when buffer usage was recorded.

(E)
: Maximum number of buffers (pages for VF and VP) at any point in time up to the point buffer usage was recorded.

(F)
: Maximum number of requests for buffers (pages for VF and VP) that were queued at any point in time up to the point buffer usage was recorded.

(G)
: Number of times the buffer (page) pool was expanded up to the point buffer usage was recorded.

(H)
: Number of buffers (pages for VF and VP) that were in the pool when buffer usage was recorded.

(I)
: Maximum number of buffers (pages for VF and VP) that were in the pool at any point in time up to the point buffer usage was recorded.

Figure 36. VTAMIO/VTAMBU Trace Record

VTAMIO Trace

The VTAMIO trace combines the following trace types:

- SVCs with codes X'31' and X'35'
- SSCH instructions
- I/O interrupts

Please find a description of the traces involved under the various trace type descriptions (SVC trace, SSCH trace and I/O trace).

SDAID Default Value

All I/O devices are traced if you do not specify any of the I/O definitions.

The format of the printed VTAMIO event records is shown in Figure 36 on page 68.

XPCC Trace

An XPCC trace provides information about connections between different applications (cross-partition communication). The information is gathered after the requested function has been processed and completed by the VSE cross-partition communication routine.

You can limit the tracing of your XPCC requests by using one or more of the XPCC trace definitions.

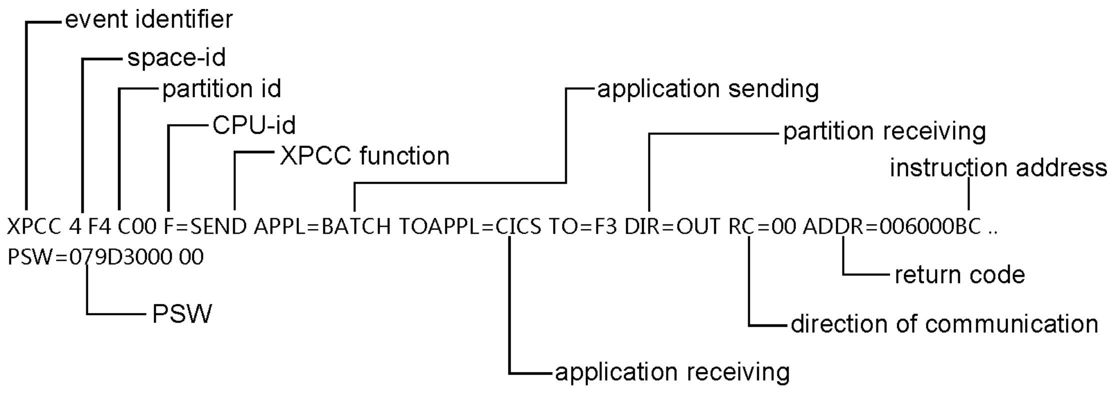
SDAID Default Value

All XPCC requests will be traced if you either:

- Do not specify one or more of the optional trace definitions.
- Code an asterisk (*) as parameter for the mandatory trace definitions.

The format of an XPCC event record is shown in Figure 37 on page 69.

XPCC Trace Output Example



```
XPCC 4 F4 C00 F=SEND APPL=BATCH TOAPPL=CICS TO=F3 DIR=OUT RC=00 ADDR=006000BC ..
PSW=079D3000 00
```

Figure 37. XPCC Trace Record

You can also display additional lines, as shown in Figure 38 on page 69.

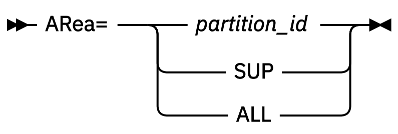
OUTPUT=XPCCB	XPCC control block
OUTPUT=XPDATABU	Data Transmit Buffer

Figure 38. Additional Fields Displayed By XPCC Trace

Notational Conventions

For a description of the syntax diagrams in this topic, see “Understanding Syntax Diagrams” on page xxiii

Defining the Area to be Traced: AREA Definition



You define the area for which processing is traced by using the AREA (or the JOBNAME) definition. You can specify either AREA or JOBNAME in the TRACE statement, but not both. For AREA, specify one of the following definitions:

- partition_id (static or dynamic)
- SUP
- ALL

Only one of these definitions is possible.

partition_id
Specifies the partition in which tasks are to be observed, like BG, F3, Y1.

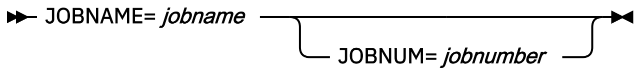
If, for example, BG is specified, the steps executed by the BG main task and by all attached subtasks are traced. ARea=BG does not necessarily mean that tracing is restricted to the BG space. For a dynamic partition (Y1, for example), the JOBNAME definition is generally used (see below).

SUP
Specifies that the activities of the supervisor are to be traced.

ALL
Specifies that the activities of all tasks in the VSE system are to be traced.

Note: For the VTAMIO trace, only ARea=partition_id can be specified, where 'partition_id' denotes the partition where VTAM is active. In that case only the I/O activities of that partition are traced.

Defining the Job to be Traced: JOBNAME Definition



If you want to trace a job in a dynamic job class, the AREA specification is not sufficient, since you do not know in which partition the job may execute. Therefore, the keywords JOBNAME and JOBNUM have been introduced which allow to trace a VSE/POWER job in a dynamic or static partition. If VSE/POWER is not installed in the system, JOBNAME is not applicable. You can specify either AREA or JOBNAME in the TRACE statement, but not both.

jobname
Specifies the name of the VSE/POWER job to be traced. Tracing starts when VSE/POWER selects the specified job for execution. The specified job name must correspond to the JNM operand of the VSE/POWER JECL statement.

jobnumber
Specifies the VSE/POWER-defined job number. It may be used if more than one job with the specified job name is contained in the VSE/POWER reader queue.

Note:

1. The JOBNAME operand cannot be used to trace a job in a VSE/POWER writer-only partition.
2. JOBNAME should not be used in a VSE/POWER-controlled partition started with the 'MT' option of the PSTART command.
3. SDAID does not accept POWER® job names containing the character '-'. (The SDAID command language uses this character as continuation character.)

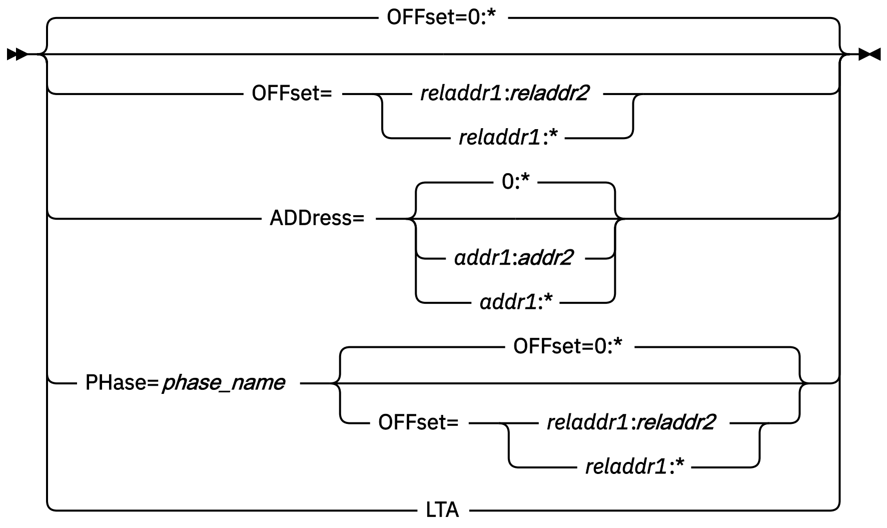
Defining the Storage to be Traced: OFFset, ADDress, PHase, LTA

Whereas the ARea and JOBNAME definitions allows you to specify the tasks that you want to observe, the storage definition determines the storage region that is to be investigated. The storage definition has to be specified in accordance to the task definition. For example, if you specified a partition-id for ARea, the definitions OFFset, ADDress, PHase, LTA can be used.

The following sections tell you the valid storage definitions in accordance with the various task specifications.

Storage Definition for AREA and JOBNAME

You may use one of the following storage area definitions for the AREA and JOBNAME definitions:



OFFset=
: limits tracing to a certain address range via offsets relative to the partition start address. OFFset=*0:** is the default specification when ARea is defined by a partition-id. The OFFSET specification is not valid for AREA=ALL.

reladdr1:reladdr2
: defines a trace address range in hexadecimal notation.

‘reladdr1’ can be any relative address within the partition, supervisor or phase defined by the ARea operand.

‘reladdr2’ must be higher than or equal to reladdr1.

reladdr1:*
: defines a trace address range starting with ‘reladdr1’ up to the end of the partition, the supervisor or the phase.

0:*
: defines that the whole storage allocated to the partition, the supervisor or the phase is defined as trace storage area.

ADDress=
: limits tracing to a certain address range within the storage allocated to VSE.

addr1:addr2
: defines a trace address range in hexadecimal notation. ‘addr1’ and ‘addr2’ can be any address in the virtual storage defined for VSE.

‘addr2’ must be higher than or equal to 'addr1'.

addr1:*
: defines a trace address range starting with ‘addr1’ up to the end of virtual storage.

0:*
: defines that the whole storage allocated to VSE is defined as trace storage area.

PHase=
: Limits tracing to a certain phase. If the named phase resides in the Shared Virtual Area (SVA), SDAID uses the start and end address of the named phase as tracing boundaries. If the named phase does not reside in the SVA, SDAID defers tracing until the named phase is loaded into a user partition.

When a load request for the named phase is issued, SDAID activates the deferred trace and updates the trace start and end address with the start and end address of the phase just loaded. This means that only those phases can be traced via the PHase parameter which (1) reside in the SVA, or (2) which are loaded into the traced partition after the STARTSD statement has been issued.

If you specify the PHase= operand, the operand OFFset= can be used in addition. In this case, OFFset= defines a trace area within the phase. If you omit the OFFset= definition, the whole storage area of the phase is traced.

reladdr1:reladdr2
defines a trace address range in hexadecimal notation.

'reladdr1' can be any relative address within the phase.

'reladdr2' must be higher than or equal to reladdr1.

reladdr1:*
defines a trace address range starting with 'reladdr1' up to the end of the phase.

0:*
defines that the whole storage allocated to the phase is defined as trace storage area.

LTA
defines that the logical transient area is specified as tracing range.

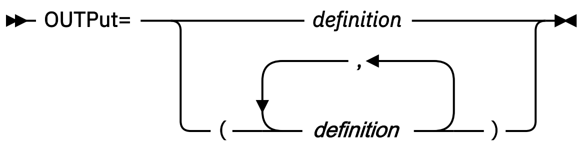
SDAID Default Values

If you do not specify OFFset, ADDress, PHase or LTA, the following defaults will be set:

- If AREA=*partition* | *supervisor*, the complete storage area that is allocated to the partition or supervisor will be traced. `OFFset=0:*` is the definition that results in direct input-mode notation.
- If AREA=ALL, or if the parameter is omitted, the complete storage area that is allocated to z/VSE will be traced.

Note: If you wish to trace in a particular partition (parameter Area=partition) and if parts of your program belong to the SVA (24-bit or 31-bit), to include this event you must explicitly increase the range of the trace using the ADDR=`0:*` parameter.

Defining Additional Trace Output: OUTPut Definition



You may specify additional trace output with the OUTPut definition. This additional trace information is recorded together with the trace event records. For example, you may define that a dump of defined control blocks or address ranges be recorded in addition to the trace event record.

You may select one or more definitions for a specific trace type.

For a summary of all OUTPut definitions, see Table 5 on page 72. The references in the third column refer you to the description of the OUTPut definitions in this topic.

Table 5. OUTPUT Definition Summary

Definition	**Records/prints in addition:**	**See:**
BUffer	Contents of SDAID output buffer	"Writing the Trace Buffer" on page 73
CCB	CCB or IORB (TRACE=IO, SSCH, or VTAMIO only)	"Recording the CCB or IORB" on page 74

Table 5. OUTPUT Definition Summary (continued)

Definition	Records/prints in addition:	See:
CCW	CCWs, IRB (TRACE=IO, SSCH, or VTAMIO only)	“Recording the CCW” on page 74
CCWD=nnnn	CCWs plus nnnn bytes of data, IRB (TRACE=IO, SSCH, or VTAMIO only)	“Recording the CCW” on page 74
COMReg	Partition communication region	“Recording the Partition Communication Region” on page 74
CReg	Control registers	“Recording the Control Registers” on page 75
DUMP	Virtual storage	“Dumping Virtual Storage” on page 75
FReg	Floating point registers	“Recording Floating-Point Registers” on page 77
GReg	General purpose and access registers	“Recording General-Purpose and Access Registers” on page 77
IOTab	PUB, LUB, ERBLOC, ERRQ, CHANQ	“Recording PUB, LUB, ERBLOC, CHANQ” on page 78
LOCKTE	Lockable entry (LOCK trace only)	“LOCK / UNLOCK Trace” on page 60
LOwcore	Processor storage from zero to X'2FF'.	“Dumping Processor Storage from X'00'to X'2FF'” on page 78
LTA	Logical transient area	“Recording the Logical Transient Area” on page 79
PTA	Physical transient area	“Recording the Physical Transient Area” on page 79
PTAB	Partition related control blocks: PCB, PIB, PIB2	“Recording Partition-Related Control Blocks” on page 79
SUP	Supervisor plus GREGs and CREGs	“Recording the Supervisor Area” on page 79
SYSCom	System communication region	“Recording the System Communication Region” on page 79
TOD	Time-of-Day clock	“Recording the Time-of-Day Clock” on page 80
TTAB	Task related control blocks: TIB, TCB, PCB, PIB, PIB2	“Recording Task-Related Control Blocks” on page 80
XPCCB	XPCC control block (XPCC trace only)	“XPCC Trace” on page 69
XPDATABU	Buffer for data to be transmitted (XPCC trace only)	“XPCC Trace” on page 69

Writing the Trace Buffer

```
►► OUTPut=BUffer ►◄
```

BUffer
writes the contents of the SDAID buffer to the device specified with the OUTDEV statement. The buffer is written immediately after the associated event has been recorded in the buffer.

Note: Specifying BUffer as one of a number of output options may result in the original event record(s) getting lost due to the wraparound recording technique used.

Recording the CCB or IORB

```
►►─ OUTPut=CCB ─►◄
```

CCB
records or prints the contents of either the CCB or the IORB (input/output request block) plus the TOD (time-of-day clock). This output option is meaningful only with an IO or SSCH trace request.

Recording the CCW

```
►►─ OUTPut= ─┬────── CCW ──────┬─►◄
             └─ (CCWD= nnnn) ──┘
```

CCW
records or prints the available channel program (Channel Command Word chain) when the trace type is SSCH. In case of an IO trace, only the CCWs which refer to transferred data are recorded or printed.

The output contains also the first 16 bytes of the associated data, the CCB, the IRB (interruption request block) and the TOD (time of day clock).

Specifying this output option for an event other than IO or SSCH is not meaningful.

CCWD=nnnnn
(CCW plus data) records or prints up to a maximum of nnnn bytes of the transferred data, the CCB, the IRB (interruption request block) and the TOD clock in addition to the information processed with the CCW specification. The number nnnn may be any (decimal) number between 1 and 65535.

The most meaningful trace type to be combined with this output option is the IO trace.

For an example of the output produced with this option, see Figure 39 on page 74.

```
I/O    S  BG   UNIT=0110            CSW =0000D610 0C000000
TOD = 95.016 17.53.54.333
CCB= 0000D2E0  00000004 0000880B 0000D5E8 0000D610


CCW= 0000D5E8  0700D47A 40000006 DATA= 0000000B 000A                                 *......          *
CCW= 0000D5F0  2300D481 40000001 DATA= 9C                                            *.               *
CCW= 0000D5F8  3100D47C 60000005 DATA= 000B000A 17                                   *.....           *
CCW= 0000D600  0800D5F8 00000000
CCW= 0000D608  8600F400 20000400


---CCW DATA---
0000F400 D7C8C1E2 C5404040 00805BD1 D6C2C3E3 D3C90020 00000000 14480048 00000000     *PHASE  ..$JOBCTLI.........*
0000F420 171C0480 00060000 00010000 16E80001 F9F4F1F1 F0F90213 717CF9F4 F1F1F1F2     *.......Y..941109...@941112*

…                         …                        …               …

----IRB-----
0001B940 00004007 0000D610 0C000000 00400000 00000000 00000000 00000000 00000000     *.. ...0...... ............*
0001B960 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00000000     *..........................*
```

Note: For a CCW with a data length less than 17 bytes data is displayed in the CCW line, else in a separate block.

Figure 39. Output of OUTPut=(CCWD=256)

Recording the Partition Communication Region

```
►►─ OUTPut=COMReg ─►◄
```

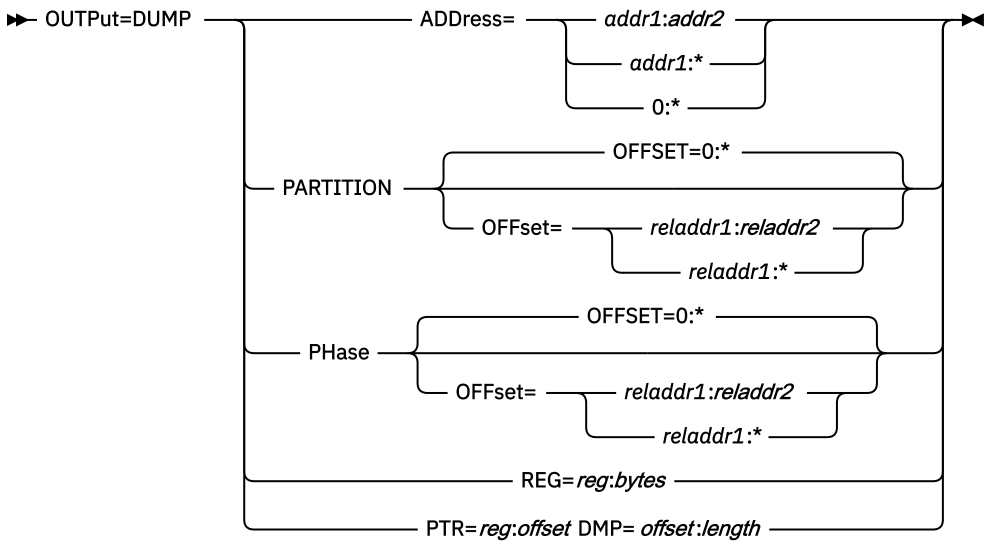
COMReg
records or prints the contents of the partition communication region.

Recording the Control Registers

```
▶▶─ OUTPut=CReg ─▶◀
```

CReg
records or prints the contents of all control registers.

Dumping Virtual Storage



Note: SDAID dumps only those areas which are in real storage, that is, accessible without page fault.

DUMP
records or prints the contents of virtual storage.

You may request up to ten different dumps.

Note: SDAID does not dump or record virtual storage above the 2 GB line.

You have to specify one or more of the dump area specifications as shown below.

PARTITION
allows to dump the partition for which the trace is active. This operand is valid for the static as well as for the dynamic partitions (AREA or JOBNAME). The OFFset operand may be defined in addition to PARTITION to limit the dump area in the partition:

reladdr1:reladdr2
defines a trace address range in hexadecimal notation.

'reladdr1' can be any relative address within the partition or the phase defined by the ARea operand.

'reladdr2' must be higher than or equal to 'reladdr1'.

reladdr1:*
defines a trace address range starting with 'reladdr1' up to the end of the partition or the phase.

0:*
defines that the whole storage allocated to the partition or the phase is defined as trace storage area.

PHase
limits the dump to the phase that is specified in the applicable area definition. The OFFset operand may be defined with the PHase operand to limit the dump area in the phase. (See above, under "PARTITION".)

ADDress

defines a dump range by a pair of addresses. For example, if you want to dump the contents of four bytes starting at storage location 0080 (hexadecimal). The definition in direct input mode would look like this:

```
ADD=0080:0083
```

addr1:addr2

defines a trace address range in hexadecimal notation. ‘addr1’ and ‘addr2’ can be any address in the virtual storage defined for VSE.

‘addr2’ must be higher than or equal to addr1.

addr1:*

defines a trace address range starting with ‘addr1’ up to the end of virtual storage.

0:*

defines that the whole storage allocated to VSE is defined as trace storage area.

REG=reg:bytes

if the starting address of the dump is specified by a register, and the number of bytes to be dumped is specified by a hex value.

The maximum number of bytes that can be specified is 1000.

PTR=reg:offset DMP=offset:length

if the dump area is located via a register plus an offset which addresses a pointer. The dump area itself is determined by the definition

```
DMP=offset:length
```

which defines the dump start address relative to that pointer and the dump length.

The offsets and the dump length are specified in hexadecimal notation. The maximum value for offset and length is 1000. The following example and Figure 40 on page 77 explain such a dump area definition.

For example:

The contents of register E is used to locate a control block. The fullword at relative offset X'10' in this control block is used as a pointer to another control block or data area.

Starting at offset X'200' an area of X'100' bytes is dumped.

```
Direct Input Mode Format

OUTPut=(DUMP PTR=E:10 DMP=200:100)
```

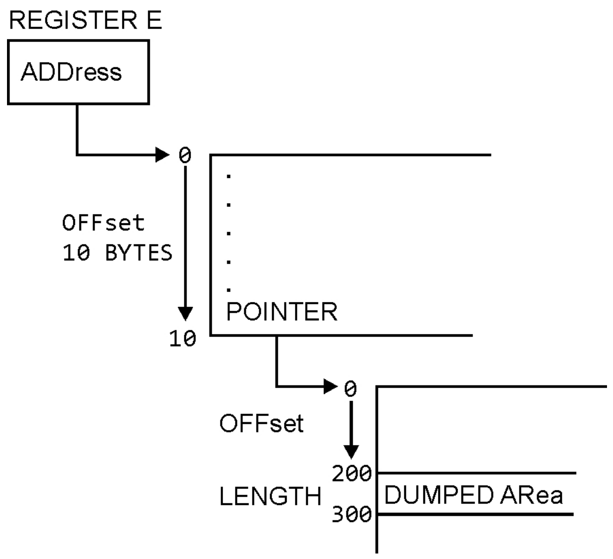


Figure 40. Overview: Defining the Area to be Dumped

In the operands REG and PTR, you can specify a general register for addressing the dump area. The PSW of the traced program determines whether the contents of the general register is interpreted as a 24-bit address or as a 31-bit address. If access registers are active, SDAID uses the general register X and the access register X to address the dumped interval.

Recording Floating-Point Registers

▶▶─ OUTPut=FReg ─▶◀

FReg

records or prints the contents of all floating-point registers. Figure 41 on page 77 shows an example of a printout of the contents of these registers.

Trace Output Example with OUTPut=FReg

```
                    ┌────────instruction that triggered the output
                    │
INST  0 BG  L  5870817C    001010B2               PSW=470D5000 001010B6
FP 0-3  4040404040404040 0000000000000000 4040404040404040 0000000000000000
   4-7  4040404040404040 0000000000000000 4040404040404040 0000000000000000
   8-B  4040404040404040 0000000000000000 4040404040404040 0000000000000000
   C-F  4040404040404040 0000000000000000 64E2D10040404040 010300FA0000A000
FPC-REG 00000000
```

Figure 41. Printout of Floating-Point Registers

Recording General-Purpose and Access Registers

▶▶─ OUTPut=GReg ─▶◀

GReg

records or prints the contents of all general-purpose and access registers (if available).

Figure 42 on page 78 and Figure 43 on page 78 show examples of printouts of the contents of the general-purpose registers, Figure 44 on page 78 of general-purpose and access registers.

Trace Output Example of General-Purpose Registers When Caller in AMODE 31

```
                           ┌───── instruction that triggered output
                           │
INST  0 BG    BALR   0577   001010B6          PSW=470D1000 0010A6D4
GR 0-7 80400C00 02021B40 80000000 FFFFFFFF 00000000 00000000 00000000 00000000
   8-F 00000000 0000000F 00100000 001A76FF 00000000 00000000 8F000000 00000200
```

Figure 42. Printout of General-Purpose Registers (AMODE 31)

Trace Output Example of General-Purpose Registers When Caller in AMODE 64

```
                           ┌───── instruction that triggered output
                           │
INST  0 BG  C00 l  5870817C   004000DC                     PSW=470D1001 804000E0
GR 0-3 0000000010FF0000 0000000000000000 0000000000000000 0000000000000000
   4-7 0000002010AF0000 0000000000000000 000000A200000000 0000000000000000
   8-B 0000100010DF0000 0000000000000000 000000000D60000 0000000000000000
   C-F 0000000010FF0000 0000000000000000 64E2D10040404040 010300FA0000A000
```

Figure 43. Printout of General-Purpose Registers (AMODE 64)

Trace Output Example with Access Registers

```
                           ┌───── instruction that triggered output
                           │
INST  0 BG    BALR   0577   001010B6          PSW=470D5000 0010A6D4
GR 0-7 80400C00 02021B40 80000000 FFFFFFFF 00000000 00000000 00000000 00000000
   8-F 00000000 0000000F 00100000 001A76FF 00000000 00000000 8F000000 00000200
AR 0-7 00000000 00000000 00000000 00000000 00000000 00000000 00000000 00020004
   8-F 00020004 00000000 00000000 00000000 00000000 00000000 00000000 00000000
```

Figure 44. Printout of General-Purpose and Access Registers

Recording PUB, LUB, ERBLOC, CHANQ

►►─ OUTPut=IOTab ─►◄

IOTab

records the contents of the following I/O tables: PUB, LUB, ERBLOC, CHANQ.

Note: If the attention routine is traced, no LUB table is recorded (only partitions have LUB tables).

Dumping Processor Storage from X'00'to X'2FF'

►►─ OUTPut=LOwcore ─►◄

LOwcore

(Lowcore contents) records or prints the contents of the first 768 bytes of processor storage (X'00' to X'2FF').

See Figure 45 on page 79 for a sample output.

Trace Output Example with OUTPut=LOwcore

```
                address of COMREG for active partition ──────┐
                                                              │
                                                              │
PGMCHK  0 BG  ????  0000    001012B8   PGCK-CODE=01      PSW=471D0000 001012BA
──LOWCORE──                                                   ▼
000000 400FA248 00000000 00001000 000FB1FA 00000000 00000330 ...
000020 47100000 00100228 471D0000 001012BA 00000000 00000000 ...
000040 00000000 00000000 000488B0 00000000 FEE67800 00000000 ...
...
```

Figure 45. Printout of Low Address Storage

Recording the Locktable Entry

►►─ OUTPut=LOCKTE ─►◄

LOCKTE
records or prints the Locktable entry of the current LOCK or UNLOCK event.

Recording the Logical Transient Area

►►─ OUTPut=LTA ─►◄

LTA
records the contents of the LTA on occurrence of the associated event.

Recording the Physical Transient Area

►►─ OUTPut=PTA ─►◄

PTA
records or prints the contents of the physical transient area on occurrence of the associated event.

Recording Partition-Related Control Blocks

►►─ OUTPut=PTAB ─►◄

PTAB
records or prints the contents of the Partition Control Block (PCB) and the Program Information Block (PIB and PIB2) of the active partition.

Recording the Supervisor Area

►►─ OUTPut=SUP ─►◄

SUP
records or prints the contents of the storage area used by the supervisor.

Recording the System Communication Region

►►─ OUTPut=SYSCom ─►◄

SYSCom
records or prints the contents of the system communication region.

Recording the Time-of-Day Clock

▶▶─ OUTPut=TOD ─▶◀

TOD
records or prints the setting of the time-of-day clock each time the associated event occurs.

Figure 46 on page 80 shows an example of this option.

Trace Output Example with OUTPut=TOD

```
                        ┌────────instruction that triggered output
                        │
                        │
PGMCHK 0 BG   ST   58B04057  001012BA  PGCK-CODE=05  PSW=471D0000 001012BE
TOD = 90.101 13.54.02.763
```

Figure 46. Program-Check Event with Time of Day

Recording Task-Related Control Blocks

▶▶─ OUTPut=TTAB ─▶◀

TTAB
records or prints the contents of the Task Information Blocks (TIBs), the Task Control Blocks (TCBs) of the tasks belonging to the active partition, and the partition-related control blocks PCB, PIB, PIB2.

Recording the XPCC Communication Control Block

▶▶─ OUTPut=XPCCB ─▶◀

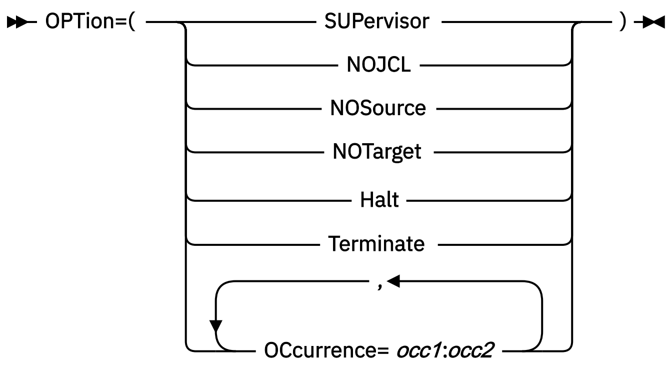
XPCCB
records or prints the contents of the XPCC communication control block.

Recording the XPCC Data Buffer

▶▶─ OUTPut=XPDATABU ─▶◀

XPDAtabu
records or prints the contents of the XPCC data buffer which is used to transfer data.

Defining the Trace Options: OPTion Definition



SUPervisor

traces a code segment within the supervisor. It allows to trace supervisor routines while they are working for a user partition (by activating the PER bit in the PSW of the supervisor routines). The specification OPTion=SUPervisor is applicable only for the branch trace, the instruction trace, and the storage alteration trace. You may use OPTion=SUPervisor if you specify AREA=partition-id or JOBNAME=jobname. Examples:

```
    TRACE INST=* AREA=F3 ADD=5000:5200 OPTION=SUP
```

This statement traces the instructions in the storage interval between 5000 and 5200 if this supervisor code is executed for the F3 partition (with the PIK of the F3 partition).

```
    TRACE BRANCH JOBNAME=TESTPROG ADD=0:* OUTPUT=GREG OPTION=SUP
```

This statement traces the effective branch instructions of the job named TESTPROG. ADD=0:* means that all storage within and outside the private user partition is to be traced. OPTION=SUPervisor causes supervisor routines to be traced while they are servicing the traced partition.

Halt

stops processing of the system when the defined trace event occurs. SDAID puts the system into a wait state with address X'00EEEE' in the address portion of the wait PSW. This wait on event enables you to perform some debugging work, for example to display registers or selected storage areas.

How to Get out of this WAIT

- Give an external interrupt.

  The trace remains initialized and the system stops at the next trace event occurrence.
- Alter storage location 0 to a value of X'FF', then give an external interrupt.

  The trace remains active but the Halt option is canceled.

NOJCL

suppresses tracing of Job Control Phases. If the keyword NOJCL is omitted, the user program and the job control statements are also traced.

NOSource

Do not record if a space switch is in effect, and the current event is within the SOURCE space.

NOTarget

Do not record if a space switch is in effect, and the current event is within the TARGET space.

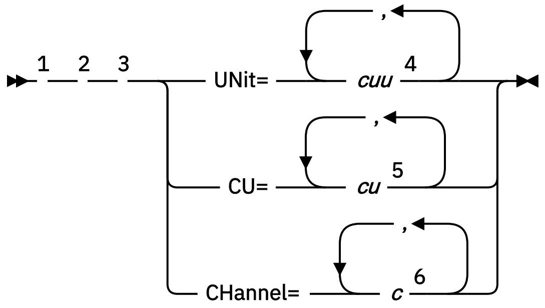
Terminate

allows you to terminate SDAID output at the occurrence of the specified event. You may start the trace output again if you issue the STOPSD and the STARTSD attention routine commands. Note, that you should issue the attention routine command ENDSD if you want to end the trace operation and release all system resources which SDAID has used.

OCcurrence=occ1:occ2

specifies the number of associated events to be traced. For example, the specification 1:20 defines that tracing starts with the first occurrence of the specified trace event and ends with occurrence 20.

Defining the Traced I/O Devices



Notes:

[1] You can specify up to eight device addresses.
[2] You can specify up to 16 control unit addresses.
[3] You can specify up to 16 channel addresses.
[4] You can specify up to eight device addresses.
[5] You can specify up to 16 control unit addresses.
[6] You can specify up to 16 channel addresses.

UNit=cuu
specifies one or up to 8 device addresses.

CU=cu
specifies one or up to 16 control unit addresses.

CHannel=c
specifies one channel address or a list of up to 16 addresses.

Note:

1. You may define these operands with the TRACE statements for the IO trace, the SSCH trace, and the VTAMIO trace.
2. The parameters UNit, CHannel, and CU are optional. If you do not specify the UNit statement, all devices are traced.
3. The parameters UNit, CHannel, and CU are mutually exclusive (in the same TRACE command).

Chapter 9. Initialize an SDAID Trace in Direct Input Mode

This topic describes how you can initialize a SDAID trace via statements read in from a SYSIN device or the attention routine (AR).

Direct input means that you enter the trace specifications directly via control statements. Beside the trace initialization in prompt mode or via procedures the direct input mode can be used to set up SDAID traces. You can use the direct input mode of SDAID to enter complete SDAID commands avoiding the time-consuming prompt input mode.

This topic contains:

Initializing an SDAID Trace in Direct Input Mode

SDAID trace initialization in direct input mode uses the following statements:

Table 6. Input Statement Summary

SDAID Statement	**See:**
EXEC SDAID or SDAID	"Starting the SDAID Trace Initialization" on page 85

Table 6. Input Statement Summary (continued)

SDAID Statement	**See:**
OUTDEV specification	“Defining the Output Device in Direct Input Mode” on page 86
TRACE type and additions	“The TRACE Statement” on page 88
/*, READY or ENTER	“Ending the SDAID Trace Initialization” on page 86

Selecting the SDAID Input Mode

In direct mode, SDAID commands can be entered either via the attention routine or via job control.

Commands Entered Via the Attention Routine:

You may initialize traces via the attention routine either in direct input mode or in prompt mode. The SDAID program may switch from one input mode to the other according to your command input. However, the statements described in this section are used for direct input mode only. The **SDAID program is started** by the attention routine command

```
SDAID
```

You **determine the direct input mode** by defining at least one keyword operand together with either the TRACE or the OUTDEV command, for example:

```
OUTDEV P=E
```

The SDAID program **switches to prompt input mode** when you omit all possible keyword operands, or if you use a prompt-input-mode statement. An example of prompt input is a statement with a question mark (?), for example:

```
TRACE INST=* AR=BG OUTP=?
```

Commands Entered Via Job Control

You may **invoke the SDAID program** in a partition using the job control statement:

```
// EXEC SDAID
```

This statement and the SDAID trace initialization commands may be entered from a console or from a SYSIN device.

You **specify the output device** of the trace with the OUTDEV statement. Only one output device specification is possible at one time in the system. Any subsequent OUTDEV statement overrides the existing one.

The TRACE statement contains the **definition of the trace type** and additional trace keyword operands. You can enter up to ten TRACE statements.

You **end the trace initialization** with the READY statement. If you entered the statements via SYSIN, the end of data (/*), or ENTER in case of console input are treated as READY statements and end the initialization process. When the READY statement has been read in, no further OUTDEV or TRACE statement can be entered.

You can **cancel the SDAID setup** during initialization with the ENDSD or CANCEL statement. Two initialization examples are shown in Figure 47 on page 85. One entered directly via the attention routine, the other read in via a SYSIN device.

In both examples the same SDAID trace is initialized.

Direct Input Setup in the Attention Routine

```
→ sdaid
  AR 015 4C05I PROCESSING OF ‘SDAID’ COMMAND SUCCESSFUL.
  AR 015 1I40I READY
→ outdev t=280
  AR 015 4C05I PROCESSING OF ‘OUTDEV’ COMMAND SUCCESSFUL.
  AR 015 1I40I READY
→ trace branch ar=bg off=0:200 outp=(greg dump add=40000:40100)
  AR 015 4C05I PROCESSING OF ‘TRACE’ COMMAND SUCCESSFUL.
  AR 015 1I40I READY
→ trace inst=(d207 95 9103) ar=bg add=0:* outp=greg
  AR 015 4C05I PROCESSING OF ‘TRACE’ COMMAND SUCCESSFUL.
  AR 015 1I40I READY
→ ready
  AR 015 4C05I PROCESSING OF ‘READY’ COMMAND SUCCESSFUL.
  AR 015 1I40I READY
```

Direct Input Setup under Control of a Partition

```
// EXEC SDAID
OUTDEV T=280
TRACE BRANCH AR=BG OFF=0:200 -
             OUTP=(GREG DUMP ADD=40000:40100)
TRACE INST=(D207 95 9103)  AR=BG ADD=0:* OUTP=GREG
/*
```

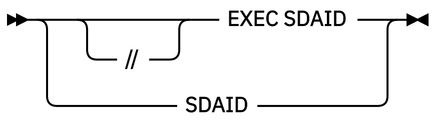
Figure 47. Trace Initialization Examples (Direct Input Mode)

Notational Conventions

- The various operands may be separated by at least one blank or by a comma.
- Enter all operands in the specified order. Examples are shown in topic “Command Input Paths” on page 136.
- For a description of the syntax diagrams please read “Understanding Syntax Diagrams” on page xxiii.
- Command continuation is allowed. It is specified by a trailing hyphen (-).
- Comments may be specified via SYSIN together with SDAID statements or as separate comment lines.

  A /* sign specifies the begin of a comment. All text from the /* sign up to the end of the line is treated as a comment. /* must not start in column 1.

Starting the SDAID Trace Initialization



You start the initialization process as follows:

- If you want to setup the trace from the AR, type only ‘sdaid’.
- If you want to initialize the trace in a partition via SYSLOG, enter ‘exec sdaid’ in that particular partition. Submit ‘exec sdaid’ or ‘// exec sdaid’ if you use SYSRDR as input device.

Ending the SDAID Trace Initialization

READY	In the Attention Routine
/*	In a Partition from SYSIPT
ENTER	In a Partition from the Console

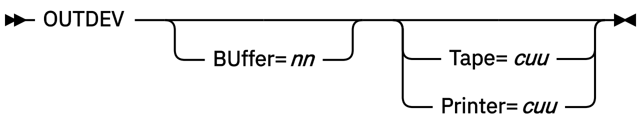
You end the initialization process with:

- The statement ‘READY’ or
- An end-of-file indication, that is:
  - **/*** from SYSIPT;
  - A blank line from the console.

Now you can start the initialized trace with the AR command STARTSD.

You find more information about starting and stopping a trace in Chapter 12, “Start/Stop and End the Trace,” on page 163.

Defining the Output Device in Direct Input Mode



You define the destination of your trace output with the OUTDEV statement.

Enter the OUTDEV statement with an operand to get into the direct input mode. Otherwise, if you entered the SDAID statement from the attention routine, the SDAID program would prompt you for the necessary information (prompt mode).

Note: An OUTDEV statement in a partition must contain at least one operand (prompt mode is not possible in a partition).

Defining the Output Device

You can define the following output destinations with the OUTDEV statement:

- Printer
- Tape
- Wraparound buffer
- Wraparound buffer and printer
- Wraparound buffer and tape

For the appropriate OUTDEV and TRACE statement for your trace initialization, see Table 7 on page 86.

Table 7. OUTDEV Summary

Device	Buffer	Output when:	SDAID Statements	Note
Printer	no	immediately	OUTDEV P=cuu	1
Tape	yes	buffer full	OUTDEV T=cuu	2
-	yes	-	OUTDEV BU=nn	3
Printer	yes	certain event occurs	OUTDEV BU=nn P=cuu TRACE type OUTP=BU	4

Table 7. OUTDEV Summary (continued)

Device	Buffer	Output when:	SDAID Statements	Note
Tape	yes	certain event occurs	OUTDEV BU=nn T=cuu TRACE type OUTP=BU	5

```
nn..... stands for the wraparound buffer size
        in units of K bytes.
        If you do not define an output device or
        you use a printer device, nn may have a
        value of 4 up to the maximum of 256.
        If you use a tape device, nn may have a
        value of 3 up to the maximum of 32.
cuu ... stands for the unit address.
        You may abbreviate it in the following way:
        00E, 0E, E
type .. represents the type of trace event which
        forces the output operation.
```

Note:

1. No buffer is allocated. The event records are printed on the printer with the device address cuu.
2. The event records are written into an internal buffer. This buffer is written to a tape mounted on the device cuu when it is full or when an ENDSD or STOPSD command is issued.
3. A trace defined with this OUTDEV statement writes the event records into a wraparound buffer. You can retrieve the trace records only with the attention routine command:

   ```
   DUMP BUFFER,cuu
   ```

   DOSVSDMP can be used to print the tape. For further information,see "Printing an SDAID or DUMP Command Produced Tape" on page 31.
4. A trace defined with this OUTDEV and TRACE statement prints the contents of the buffer when the trace event (type), defined with the TRACE statement, occurs.
5. The defined wraparound buffer is written to tape (cuu) when the trace event (type) occurs.

Note: From z/VSE 3.1 onwards, the buffer is allocated in the 31-bit SVA. It is no longer part of the SDAID area. The buffer file is rounded up to a multiple of 4K bytes.

SDAID Trace Initialization Example

Assume you want to use SDAID to trace instructions executed in the BG partition. The buffer should be 6K to allow for a reasonable number of trace event records, and should be written to tape when end-of-job is reached or the partition is canceled. Figure 48 on page 87 shows the statements necessary to initialize this trace. The SDAID statements are entered in direct input mode via SYSIN.

```
// EXEC SDAID
OUTDEV BU=8 T=280
TRACE INST=* AR=BG
TRACE CA AREA=BG OUTP=BU,OPT=TERM
/*
```

Figure 48. Example: Initializing an SDAID Trace

For the trace initialized in Figure 48 on page 87:

- The buffer size is 8K bytes (OUTDEV BU=8 ...);
- The output tape is mounted on unit address 280 (OUTDEV .... T=280);
- SDAID traces instructions (TRACE INST=* ...) executed in the BG partition (... AR=BG) into the buffer;
- The buffer is written to a tape when a CANCEL or an EOJ condition is encountered (TRACE CA OUTP=BU ...);
- Tracing stops after the buffer is written (... OPT=TERM).

The TRACE Statement

The TRACE statement is used to define the trace types you need to get the most reasonable information about errors in your computing environment. For the appropriate trace type, see Table 8 on page 88.

This section describes the format of all SDAID trace type initializations in direct input mode and shows trace statement and trace initialization examples.

The possible abbreviations are shown through lowercase letters.

Trace statements can be entered up to column 72 of the input line, and may be continued on the following line or lines. To continue a statement, enter at least one blank character and a hyphen (-) after the last operand in the first line, and continue the statement between columns 1 and 72 of the following line.

Enter all operands in the specified order. Refer to the examples in topic "Command Input Paths" on page 136.

Summary of Trace Types

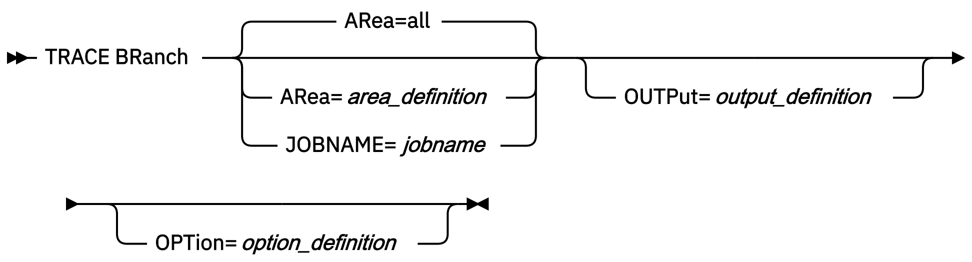
Table 8. Trace Type Summary

Trace Type	**Provides a Trace of:**	**See:**
BRanch	Successfully executed branch instructions	"BRanch Trace" on page 89
BUffer	The trace buffer when it is full	"BUffer Trace" on page 89
CAncel	Program (main task) cancel or EOJ	"CAncel Trace" on page 90
EXTernal	External interrupts	"EXTernal Trace" on page 91
GETVIS	Getvis / Freevis requests	"GETVIS Trace" on page 92
INSTruction	Selected or all instruction(s) execution	"INSTruction Trace" on page 94
IO	I/O interrupts	"I/O Interrupt Trace" on page 95
LOCK	Lock / unlock requests of resources	"LOCK Trace" on page 96
MONitorcall	MC instructions	"MONitor Call Trace" on page 98
OSAX	OSAX adapter	"OSAX Adapter Trace" on page 99
PGMCheck	Program checks	"PGMCheck Trace" on page 100
PGMLoad	Phase load requests, or actual load	"Program Load Trace (Fetch/Load Trace)" on page 101
SSCH	Start Subchannel instructions	"Statement Examples" on page 103
STorage	Storage alterations	"Storage Alteration Trace" on page 103
SVC	Executed supervisor calls	"Supervisor Call Trace" on page 105
VTAMBU	Usage of VTAM buffers	"VTAM BUffer Trace" on page 106
VTAMIO	VTAM I/O operations	"VTAMIO Trace" on page 107
XPCC	XPCC communication requests	"XPCC Trace" on page 108

Besides the type of the trace and some definitions which belong to the trace type, other keyword operands like AREA, OUTPUT or OPTION may be defined to limit the trace or to produce additional trace output.

These other operands are grouped together and their format is shown under "Additional Definitions" on page 110.

BRanch Trace



Note: For performance reasons, ARea=all requires a specification of a limited address range via the ADDRESS parameter.

For an explanation of:

- area_definition, see "ARea or JOBNAME Definition" on page 111.
- jobname, see "ARea or JOBNAME Definition" on page 111.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

For a description of the trace and an example of the output, see "BRANCH Trace" on page 56.

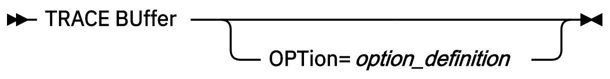
Initialization Example

```
// EXEC SDAID
OUTDEV T=280
TRACE BR AR=BG ADDR=0:* -
         OPT=(NOJCL,SUP)
/*
```

The following items are covered by the trace set up as shown in the example.

- Use the SYSRDR device to set up the trace.
- Direct the output of the trace data to the tape at device address 280.
- The branch instructions under the control of the BG partition are to be traced.
- The whole system is to be observed to record the instructions executed for the BG partition, including supervisor routines working for BG.
- Do not trace the job control branch instructions.

BUffer Trace



For an explanation of option_definition, see "OPTion Definition" on page 116.

For a description of the trace, see "BUFFER Trace" on page 57.

The BUffer trace output is the collection of all trace records contained in the buffer when a buffer overflow occurs.

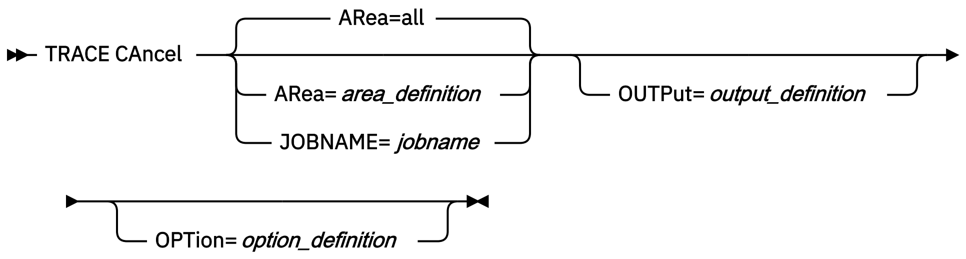
Initialization Example

```
// EXEC SDAID
OUTDEV BU=6 T=280
TRACE BR AR=BG ADDR=0:* -
         OPT=NOJCL
TRACE BU
/*
```

The following items are covered by the trace set up as shown in the example.

- Use the SYSRDR device to set up the trace.
- Trace all successfully executed branch instructions.
- The branch instructions of the BG tasks are to be traced.
- Observe the whole storage.
- Collect the trace data in a 6K byte buffer.
- Output the buffer whenever it is full.
- Write the output to the tape on device address 280.
- Do not trace the job control branch instructions.

CAncel Trace



Note: The parameters ADDRESS, OFFSET, PHASE and LTA are not applicable for the CANCEL trace.

For an explanation of:

- area_definition, see "ARea or JOBNAME Definition" on page 111.
- jobname, see "ARea or JOBNAME Definition" on page 111.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

For a description of the trace and an example of the output, see "CANCEL Trace" on page 57.

Statement Example

```
TRACE CA AR=BG -
         OUTP=BU
```

The trace statement shown in the example initializes a CAncel trace which writes an event record and the buffer whenever a cancel or EOJ condition occurs in the BG partition.

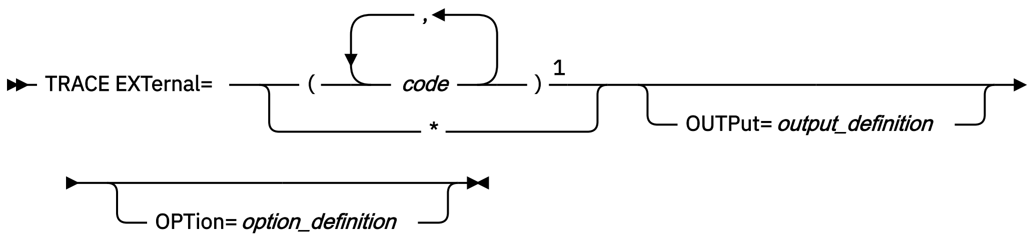
Initialization Example

```
// EXEC SDAID
OUTDEV P=E
TRACE CA AR=BG -
         OUTP=(DUMP PART OFF=78:1000)
/*
```

The following items are covered by the trace setup shown:

- Use the SYSRDR device to set up the trace.
- Trace cancel and EOJ conditions.
- Print the area between BG relative address X'78' and X'1000' together with the event record.
- Output the trace record on the printer at device address 00E.

EXTernal Trace



Notes:

[1] Up to eight codes may be specified.

For an explanation of:

- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

'code' may be one to 8 of the following:

```
0040   Interrupt key
1003   TOD-clock sync check
1004   Clock comparator
1005   CPU timer
1200   Malfunction alert
1201   Emergency signal
1202   External call
2401   Service signal
2402   Logical device               * z/VM CP
2603   PFAULT handshaking           * z/VM CP
4000   IUCV, APPC                   * z/VM CP
4001   VMCF                         * z/VM CP
```

`EXTernal=*` traces all external interruptions.

If you specify more than one external interrupt type, separate them by one or more blanks or by a comma (with or without blanks) and enclose them in brackets.

For a description of the trace and an example of the output, see "EXTERNAL Trace" on page 57.

Statement Example

```
TRACE EXT=0040 -
         OUTP=(TOD,BU)
```

The example shows an external interrupt trace. 0040 is defined as external interrupt type. This interrupt is used to have the wraparound buffer and the TOD clock recorded or printed together with the external trace event record.

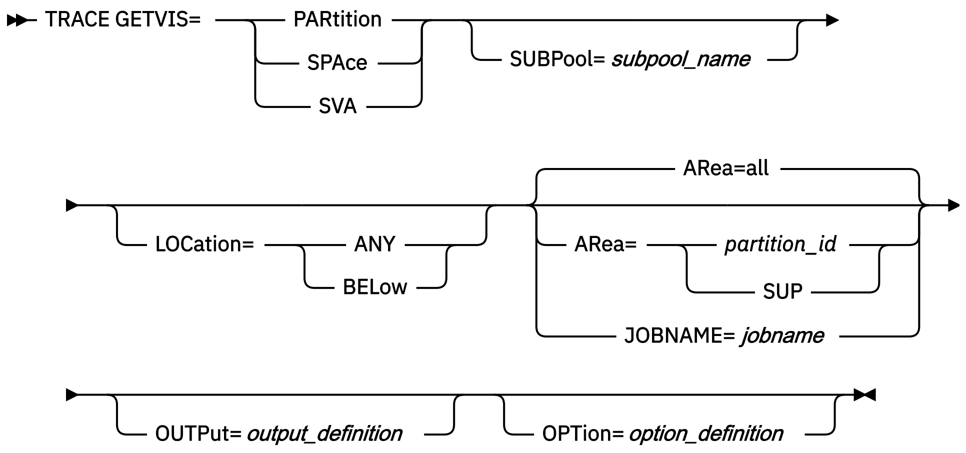
Initialization Example

```
// EXEC SDAID
OUTDEV T=280
TRACE EXT=(2401 1005) -
      OUTP=TOD
/*
```

The following items are covered by the trace setup shown:

- Use the SYSRDR device to set up the trace.
- Trace all signal and timer interrupts.
- Output the trace data to the tape on device address 280.
- Add the TOD clock to the trace data output.

GETVIS Trace



For an explanation of:

- jobname, see "ARea or JOBNAME Definition" on page 111.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

GETVis=PARtition | SPAce | SVA
GETVis=PARtition traces all Getvis or Freevis requests within the Partition Getvis area.

GETVis=SPAce traces all Getvis or Freevis requests within the Space Getvis area of a dynamic partition. **Note:** If you issue a Space Getvis request for a *static* partition, the required space will be allocated in the SVA.

GETVis=SVA traces all Getvis or Freevis requests within the Shared Virtual Area (SVA).

SUBPool=*subpool_name* | *nnn | *nnn<hex>* | DEFAULT**
A subpool_name may consist of up to six characters.

- If you specify a subpool name, Getvis or Freevis requests will be traced that are within the specified subpool.
- If you do not specify a subpool name, *all* Getvis or Freevis requests will be traced.

You can enter a subpool as one of the following:

subpool_name
Printable characters (EBCDIC) which must follow the naming conventions described in the Getvis / Freevis macros.

nnn*
If a character string is followed by an asterisk (*), all requests will be traced to those subpools whose names begin with the character string.

nnn<hex>
A subpool name may be a mixture of printable and hexadecimal characters. Characters enclosed in angled brackets **<** and **>** will be taken as hexadecimal characters. All requests will be traced to those subpools whose names have this format. Here are some examples of mixed printable and hexadecimal characters:

- SUBP=INLC<21> refers to a subpool with an internal representation of C9D5D3C321.
- SUBP=INLC21 refers to a subpool with an internal representation of C9D5D3C3F2F1.

DEFAULT
Causes all requests to the common default subpool to be traced. This default subpool is used if you do not specify a subpool_name in the GETVIS invocation macro.

Note: DEFAULT is accepted although it has a length that is greater than six characters. This is because DEFAULT is treated as a keyword and not as a subpool name.

LOCation=BELow | ANY
LOCation can take one of two values:

LOCation=BELow
Causes only those Getvis / Freevis requests to be traced that are within the 24-bit Getvis area or within the 24-bit SVA.

LOCation=ANY
Causes *all* Getvis / Freevis requests to be traced that are within the 24-bit or 31-bit areas.

ARea=*partition_id* | SUP | ALL
Causes Getvis / Freevis requests to be traced for tasks running within the specified area. If you omit the ARea operand, all Getvis / Freevis requests will be traced that are executed within the system.

For a description of the trace and an example of the output, see "GETVIS / FREEVIS Trace" on page 58.

Statement Examples

```
 TRACE GETVIS=PAR SUBP=MYPOOL AREA=BG
 TRACE GETVIS=SVA LOC=BEL
 TRACE GETVIS=SVA SUBP=DEFAULT AREA=BG
 TRACE GETVIS=PAR SUBP=INLC<00>
```

In the above four examples, the TRACE statements shown:

1. Trace all BG requests within the subpool MYPOOL in the BG partition Getvis area.
2. Trace all system-wide requests within the 24-bit SVA.
3. Trace all BG requests within the common system default subpool.
4. Trace all system-wide requests within the subpool INLC00, where "00" is treated as a hex character.

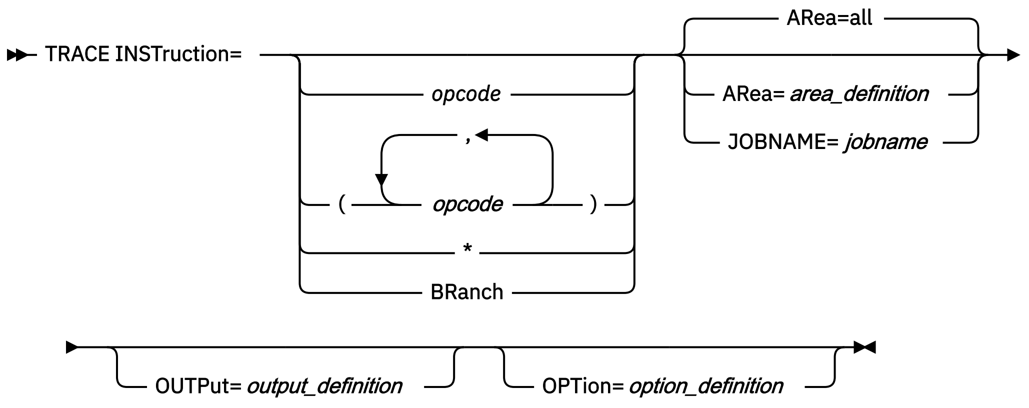
Initialization Example

```
// EXEC SDAID
OUTDEV T=280
TRACE GETVIS=PAR AR=BG OPT=NOJCL
/*
```

The following items are covered by the trace setup shown:

- Use the SYSRDR device to set up the trace.
- Trace all partition Getvis requests for BG.
- Output the trace data to the tape on device address 280.
- Do not trace the job control Getvis requests.

INSTruction Trace



Note: For performance reasons, ARea=all requires a specification of a limited address range via the ADDRESS parameter.

For an explanation of:

- area_definition, see "ARea or JOBNAME Definition" on page 111.
- jobname, see "ARea or JOBNAME Definition" on page 111.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

opcode
(one to eight) entered as either one-byte or two-byte hexadecimal instruction codes.

If you specify more than one operation code, separate them by one or more blanks or by a comma (with or without blanks) and enclose them in brackets.

*** (asterisk)**
requests a trace of *all* executed instructions.

BRanch
requests that all types of branch instructions be recorded.

For a description of the trace and an example of the output, see "INSTRUCTION Trace" on page 59.

Statement Examples

```
TRACE INST=D7 AR=BG
TRACE INSTR=* AR=BG ADD=0:*
TRACE INST=BR AR=BG
TRACE INST=(92 D204) AR=BG
```

The statements shown:

- Trace CLC instructions in BG partition
- Trace all BG task instructions
- Trace branch type instructions in BG partition
- Trace selected instructions in BG partition.

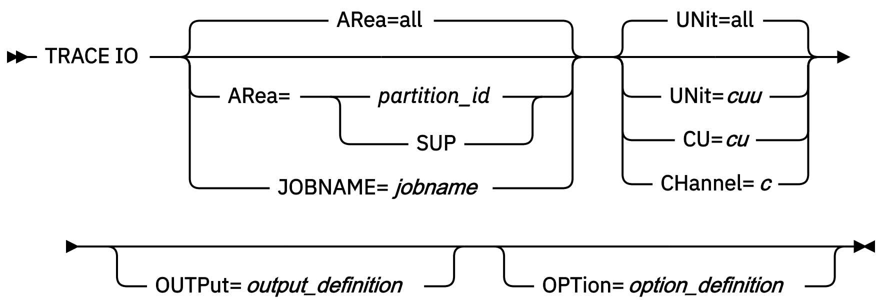
Initialization Example

```
// EXEC SDAID
OUTDEV BU=4 T=280
TRACE INST=D2 AR=BG -
         OPT=NOJCL
TRACE PGMC=* AR=BG ADD=0:* -
         OUTP=BU
/*
```

The following items are covered by the trace setup shown:

- Use the SYSRDR device to set up the trace.
- Collect all trace data into a wraparound buffer.
- The MVC instructions of BG tasks are to be traced.
- Write the buffer to the tape on device address 280 when a program check occurs in the BG partition.
- Do not trace the job control MVC instructions.

I/O Interrupt Trace



For an explanation of:

- area_definition, see "ARea or JOBNAME Definition" on page 111.
- jobname, see "ARea or JOBNAME Definition" on page 111.
- cuu, cu and c, see "I/O Device Definition" on page 113.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

Note that for the area and jobname definitions, the parameters ADDress, OFFset, PHase, and LTA are not applicable.

For a description of the trace and an example of the output, see "IO Trace (I/O Interrupt)" on page 59.

Statement Examples

```
TRACE IO  UNIT=(130 133) -
          OUTP=(CCWD=512)

TRACE IO  CU=28 OUTP=CCW
```

The TRACE statements shown in the example define additional trace output (CCWD=512 and CCW) to the normal IO trace event records.

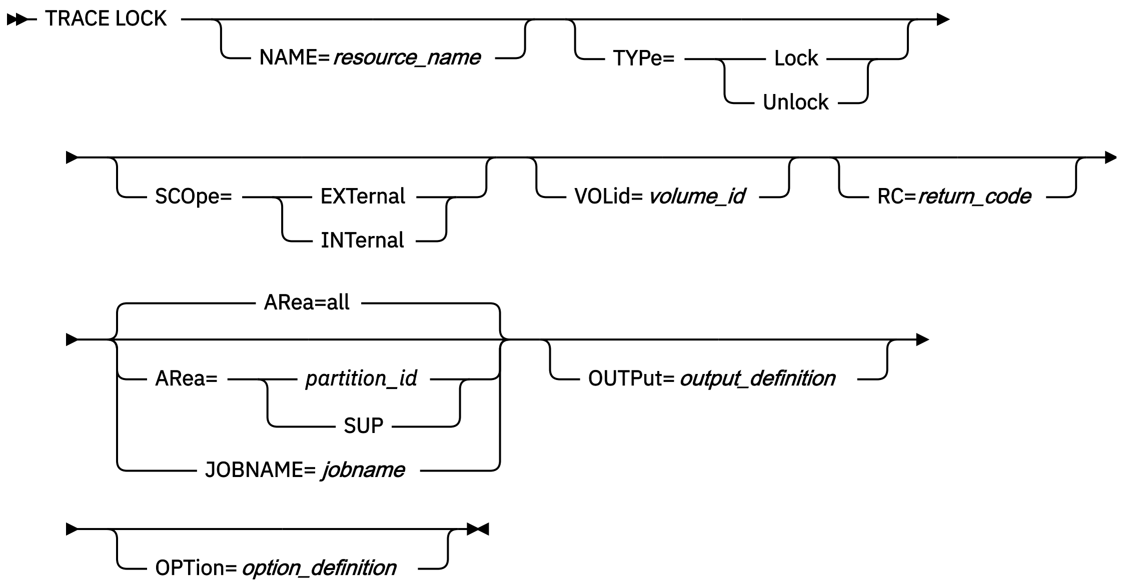
Initialization Example

```
// EXEC SDAID
OUTDEV T=280
TRACE IO -
      UN=320 -
      OUTP=CCWD=512
/*
```

The following items are covered by the trace setup shown:

- Use the SYSRDR device to set up the trace.
- Trace all I/O interrupts from device with the address 320.
- Output the trace data to the tape on device address 280.
- Add the CCW plus up to 512 bytes of transferred data to the trace event record.

LOCK Trace



For an explanation of:

- jobname, see "ARea or JOBNAME Definition" on page 111.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

NAMe=*resource_name* | *nnn | *nnn<hex>***

A resource name may consist of up to twelve characters.

- If you specify a resource name, lock or unlock requests will be traced that are related to the specified resource.
- If you do not specify a resource name, *all* lock or unlock requests will be traced.

You can enter a resource name as one of the following:

resource_name

Printable characters (EBCDIC) which must follow the naming conventions described in the DTL invocation macros.

nnn*

If a character string is followed by an asterisk (*), all requests will be traced to those resources whose names begin with the character string.

nnn<hex>

A resource name may be a mixture of printable and hexadecimal characters. Characters enclosed in angled brackets **<** and **>** will be taken as hexadecimal characters. All requests will be traced to those resources whose names have this format. Here are some examples of mixed printable and hexadecimal characters:

- NAME=INLC<21> refers to a resource with an internal representation of C9D5D3C321.
- NAME=INLC<21> refers to a resource with an internal representation of C9D5D3C3F2F1.

TYPe=Lock | Unlock

Type can take one of two values:

TYPe=Lock

Causes only the *locking* of one or more resources to be traced.

TYPe=Unlock

Causes only the *unlocking* of one or more resources to be traced.

If this parameter is omitted, both locking and unlocking of one or more resources will be traced.

SCOpe=EXTernal | INTernal

Scope can take one of two values:

SCOpe=EXTernal

Causes only *external* locks to be traced.

SCOpe=INTernal

Causes only *internal* locks to be traced.

If this parameter is omitted, both external and internal locks will be traced.

VOLid=*volume_id*

A six-character volume ID. Only events related to the specified volume_id will be traced. If the parameter is omitted, all volume IDs will be traced.

Note: The parameters scope and volume ID are mutually exclusive. This means, you can only specify one of these parameters.

RC=*nn* | >*nn* | <*nn* | (*nn ...*)

Return code can take one of four values:

RC=*nn*

Causes all events with a final return code equal to the specified return code to be traced.

RC=>*nn*

Causes all events with a final return code greater than the specified return code to be traced.

RC=<*nn*

Causes all events with a final return code lower than the specified return code to be traced.

RC=(*nn ...*)

You can specify up to sixteen return codes within parentheses. This causes all events with a final return code equal to one of the specified return codes to be traced.

ARea=*partition_id* | SUP | <u>ALL</u>

Causes lock / unlock requests to be traced for tasks running within the specified area. If you omit the ARea operand, *all* lock / unlock requests will be traced that are executed within the system.

For a description of the trace and an example of the output, see "LOCK / UNLOCK Trace" on page 60.

Statement Examples

```
TRACE LOCK
TRACE LOCK NAME=MYRESOURCE TYPE=LOCK
TRACE LOCK RC=>0 SCOPE=INT AREA=BG
TRACE LOCK VOLID=SHARE1
```

In the above four examples, the TRACE statements shown:

1. Trace all lock / unlock events.
2. Trace all lock events for resource MYRESOURCE.
3. Trace all internal BG lock / unlock events with a final return code greater than zero.
4. Trace all lock / unlock events related to VOLID=SHARE1.

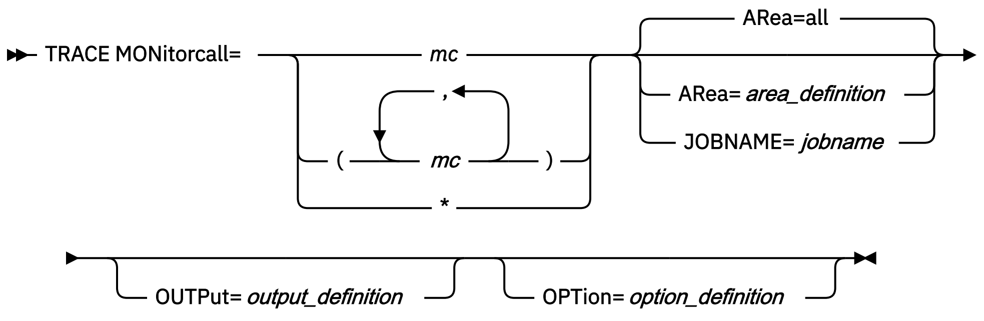
Initialization Example

```
// EXEC SDAID
OUTDEV T=280
TRACE LOCK AR=BG OPT=NOJCL
/*
```

The following items are covered by the trace setup shown:

- Use the SYSRDR device to set up the trace.
- Trace all lock / unlock events for BG.
- Output the trace data to the tape on device address 280.
- Do not trace the job control Getvis requests.

MONitor Call Trace



For an explanation of:

- area_definition, see "ARea or JOBNAME Definition" on page 111.
- jobname, see "ARea or JOBNAME Definition" on page 111.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

mc (monitor classes)
defines the monitor class of the MC instructions to be traced. Only the monitor call instructions with the defined class are traced. Up to eight classes may be defined.

The monitor classes must be specified as one-digit hexadecimal values. If you specify two or more classes, they must be enclosed in brackets and separated by one or more blanks, or by a comma with or without one or more blanks.

You may specify any valid monitor class; however, SDAID ignores a specification of class 2, because class 2 is used by SDAID to control tracing.

*** (asterisk)**
provides an event record for any execution of an MC instruction (except an MC instruction with class 2 specified) within the range of the trace operation.

For a description of the trace and an example of the output, see "MONITORCALL Trace" on page 61.

Statement Examples

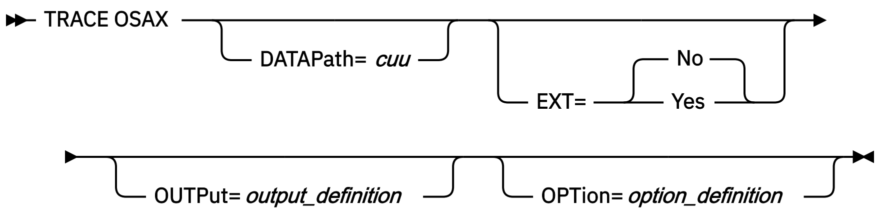
```
TRACE MON=3 AREA=ALL

TRACE MON=* AREA=ALL

TRACE MON=(3 4) AREA=ALL
```

The statements shown:

- Trace all class 3 monitor calls;
- Trace all monitor calls;
- Trace all class 3 and 4 MCs.

OSAX Adapter Trace

Note: the output from an OSAX adapter trace is complex and normally only suitable for use by IBM personnel. Typically, an OSAX adapter trace would be requested by IBM support personnel after a related problem has been reported to them.



For an explanation of:

- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

DATAPath=*cuu*
defines the data path address of an OSAX adapter. If this parameter is omitted, the default is to trace all data paths.

EXT=Yes|No
requests additional detailed output for the trace. The default is that additional output will **not** be produced.

For a description of the trace and an example of the output, see "OSAX Adapter Trace" on page 62.

Statement Examples

```
TRACE OSAX

TRACE OSAX DATAP=A0D

TRACE OSAX DATAP=A0D EXT=Y
```

The statements shown will:

- Trace all data path addresses;
- Trace the data path address A0D;
- Trace the data path address A0D but with additional detailed output;

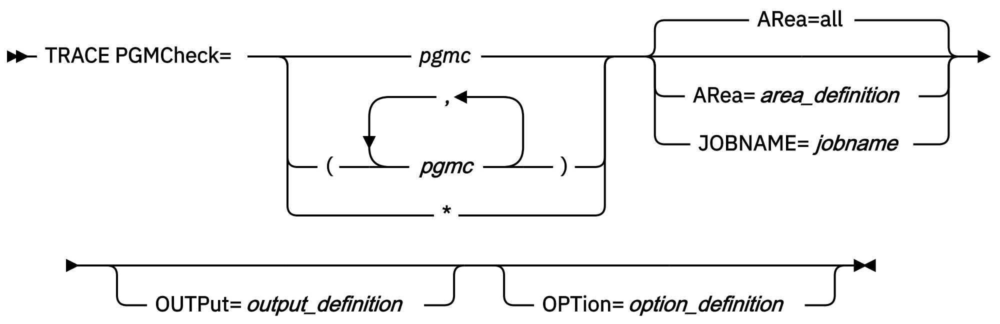
Initialization Example

```
// EXEC SDAID
OUTDEV T=280
TRACE OSAX DATAP=A0D OUTP=GREG
/*
```

The following items are covered by the trace set up as shown in the example:

- Use the SYSRDR device to set up the trace;
- Direct the output to the tape on device address 280;
- Trace the data path A0D;
- Record GREGS as additional output for every trace event;

PGMCheck Trace



For an explanation of:

- area_definition, see "ARea or JOBNAME Definition" on page 111.
- jobname, see "ARea or JOBNAME Definition" on page 111.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

pgmc
: At least one program interruption code (up to 16) must be specified in hexadecimal notation; leading zeros may be omitted.

  If you specify more than one program interruption code, they must be enclosed in parentheses and separated by one or more blanks, or by a comma with one or more blanks.

*** (asterisk)**
: requests a trace of all valid program interrupt codes with a value less than X'40', except those page or segment translation exceptions which are caused by the temporary absence of a storage page. The specification PGMC=(10 11) traces all page or segment translation exceptions.

For a description of the trace and an example of the output, see "PGMCheck Trace (Program Check)" on page 63.

Statement Examples

```
TRACE PGMC=5 AR=BG

TRACE PGMC=* AR=BG ADD=0:*

TRACE PGMC=(1 A 11) AR=BG
```

The statements shown:

- Trace program check addressing exceptions in BG partition;

- Trace all program checks of BG tasks;
- Trace the program checks of BG partition with interruption codes:
  - 1 ... operation exception;
  - A ... decimal overflow exception;
  - 11 ... page translation exception.

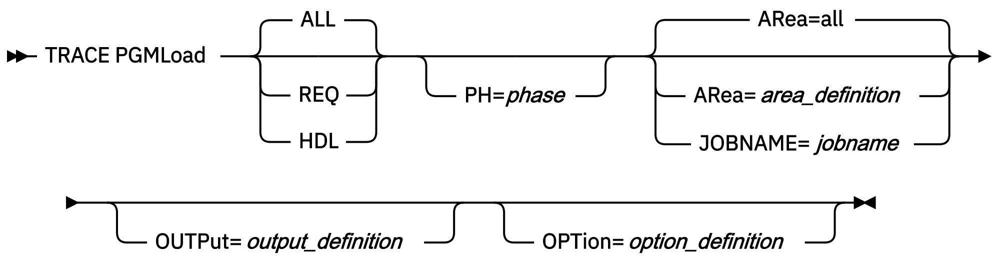
Initialization Example

```
// EXEC SDAID
OUTDEV BU=4 T=280
TRACE INST=D2 AR=F2 -
         OPT=NOJCL
TRACE PGMC=1 AR=F2 ADD=0:* -
         OUTP=BU
/*
```

The following items are covered by the trace set up as shown in the example:

- Use the SYSRDR device to set up the trace;
- Direct the output to the tape on device address 280;
- Trace the MVC instructions (D2) of the F2 partition;
- Collect the trace event records in a 4K bytes wraparound buffer.
- Write the buffer to the output tape when a program check interrupt with interruption code 'operation exception (0001)' occurs in the F2 partition.
- Do not trace the instructions executed during job control processing.

Program Load Trace (Fetch/Load Trace)



For an explanation of:

- area_definition, see "ARea or JOBNAME Definition" on page 111.
- jobname, see "ARea or JOBNAME Definition" on page 111.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

REQ
defines, that an event record for each request for fetching/loading a phase is to be written.

HDL
defines that an event record is to be written each time a phase fetch/load request is handled; that is, when a requested phase is actually loaded into storage for execution.

ALL
combines REQ and HDL. This is the default.

PH=phase
defines the phase whose program load events should be traced.

If the ARea definition is included, only the following ADDress definition is allowed (without OFFset, PHase, LTA):

ADDress=addr1:addr2|addr1:*|0:*
: defines that only program load events occurring within the specified address range are to be traced.

  SDAID records the SVCs issued within the address range and those load completion events that occur if the phase is loaded into the specified address range.

For a description of the trace and an example of the output, see "PGMLOAD (Fetch/Load) Trace" on page 64.

Statement Examples

```
TRACE PGML AR=BG
TRACE PGML PH=PROGR1 AR=F2
```

The statements shown:

- Trace all program load events for BG tasks;
- Trace all F2 task program load events for the phase PROGR1;

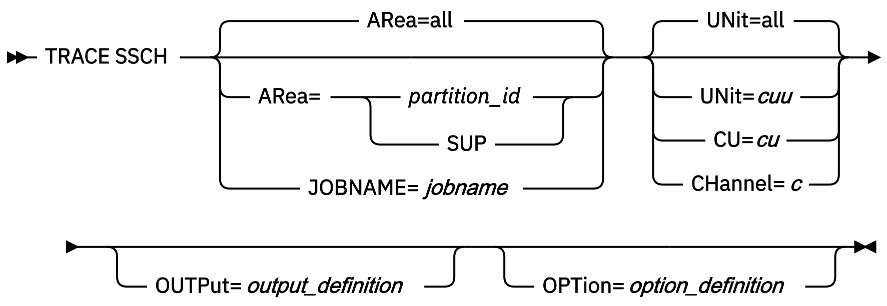
Initialization Example

```
// EXEC SDAID
OUTDEV BU=6 T=280
TRACE PGML HDL AR=BG -
      OUTP=(DUMP PART OFF=1000:2000) -
      OPT=NOJCL
TRACE PGMC=1 AR=BG ADD=0:* -
      OUTP=(DUMP PART OFF=1000:2000 -
      BUFFER) -
      OPT=NOJCL
/*
```

The following items are covered by the trace setup as shown in the example:

- Use the SYSRDR device to set up the trace;
- Trace all fetch/load executions of the BG partition;
- Record the trace data in a 6K bytes wraparound buffer;
- Write the trace data to the output tape at device address 280 when a program check operation exception (interrupt code 1) occurs in the BG partition;
- Add a dump of the BG area between relative address 1000 to 2000 to both event records;
- Do not trace the job control activities.

SSCH Instruction Trace



For an explanation of:

- jobname, see "ARea or JOBNAME Definition" on page 111.

- UNit=page, see "I/O Device Definition" on page 113.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

ARea=partition-id|SUP|ALL
: causes SSCH instructions of tasks running in the specified area to be traced. Only the specifications shown above are possible. If you omit the ARea operand, all SSCH instructions executed in the system will be traced.

UNit,CU,CHannel
: limits the trace to SSCH instructions to a certain unit, control unit or channel. If you omit these operands, no device address limitation is used.

For a description of the trace and an example of the output, see "SSCH Instruction Trace" on page 65.

Statement Examples

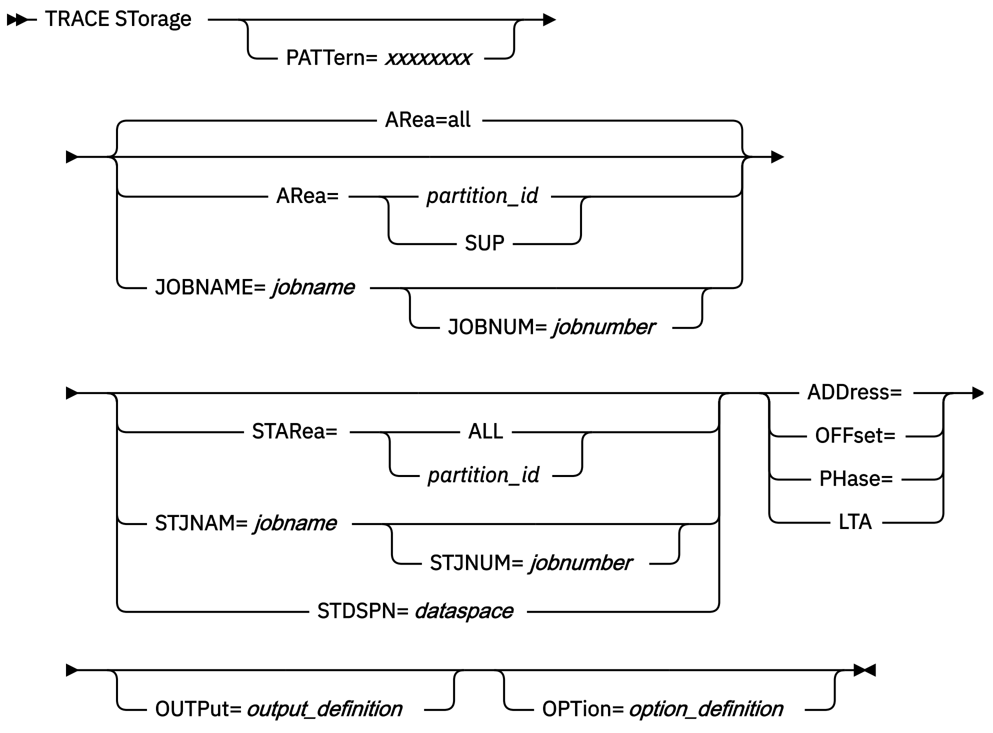
```
(1)   TRACE SSCH AR=F4 UNIT=009 -
                 OUTPUT=CCB

(2)   TRACE SSCH CHANNEL=(2 3) -
                 OUTP=TOD
```

The TRACE statements in the example define the following functions:

(1)
: Trace each SSCH instruction of F4 tasks for the device with the device address 009. Add a dump of the CCB to each SSCH trace event record.

(2)
: Trace all SSCH instructions which concern the channels 2 and 3. Add the time of day entry (TOD) to each SSCH trace event record. You will find an example of such a TOD entry under "Trace Output Example with OUTPut=TOD" on page 80.

Storage Alteration Trace



Note: For performance reasons, ARea=all requires a specification of a limited address range via the ADDRESS parameter.

For an explanation of:

- area_definition, see "ARea or JOBNAME Definition" on page 111.
- jobname, see "ARea or JOBNAME Definition" on page 111.
- address-, offset-, and phase-definition, see "ADDress Definition" on page 111.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

The storage alteration trace monitors instructions which alter a specified storage location. The altering program (source of alteration) and the storage area to be altered (target of alteration) may be in the same space or in a different space. A program running in the primary address space A can alter a storage area in the primary space A, or in an address space B, or in a data space C. The keywords AREA=partition-id|SUP|ALL and JOBNAME specify the tasks which alter a storage location (source of alteration). The keywords STAREA=partition-id|ALL, STJNAM, and STDSPN specify the target space where storage alteration is to be monitored. The keywords ADDRESS, OFFSET, PHASE, and LTA specify the target address.

PATTern=xxxxxxxx
defines a hexadecimal storage pattern up to four bytes long.

If you specify an odd number of digits, a zero is inserted to the left of the first specified hexadecimal digit.

ARea=
JOBNAME=
defines those tasks whose alteration activities you want to trace. If you do not know which task does the alteration, specify ARea=ALL to have all tasks of the system watched.

STARea=ALL
specifies a storage alteration within any address space or data space.

STARea=partition-id
specifies a storage alteration in the private address space where the named partition is allocated.

STJNAM=jobname[STJNUM=jobnumber]
specifies a storage alteration in the private address space where the named POWER job (with the named job number) executes. Note that SDAID does not accept POWER job names containing the character '-'. (The SDAID command language uses this character as a continuation character.)

STDSPN=dataspace
specifies a storage alteration within the specified data space.

If none of the keywords STARea, STJNAM, STJNUM, and STDSPN is specified, the corresponding ARea, JOBNAME, and JOBNUM keywords apply.

ADDress= | OFFset= | PHase= | LTA
specifies the address (or offset, phase, LTA) of the **target** area (where the alteration takes place).

Do not use the definition OFFset if STARea=ALL or STDSPN= is specified. Do not use the definition PHase or LTA if STDSPN= is specified.

For a description of the trace and an example of the output, see "STORAGE Alteration Trace" on page 66.

Statement Example

```
TRACE ST PATT=D205 -
         AR=ALL -
         ADD=65674:65675
```

The example records all instructions which alter the contents of two bytes starting with storage location X'65674' to the pattern X'D205'.

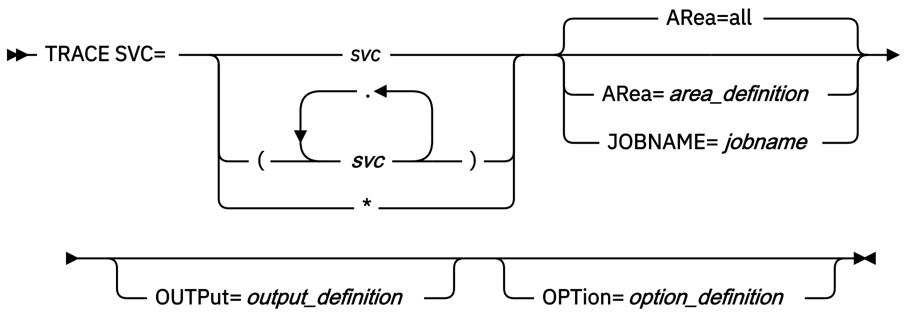
Initialization Example

```
// EXEC SDAID
OUTDEV BU=3 T=280
TRACE INST=* AR=BG -
             OUTP=GREG -
             OPT=NOJCL
TRACE ST PATT=FFFF -
             AR=ALL -
             ADD=40100:40101 -
             OUTP=BU
/*
```

The following items are covered by the trace setup shown:

- Use the SYSRDR device to set up the trace.
- Trace all BG instructions excluding the job control instructions.
- Record the event records in a 3K bytes wraparound buffer.
- Write the buffer together with a storage alter trace event record to the tape on device address 280 when the storage area with the address 40100 to 40101 is altered to X'FFFF'.
- Observe all tasks of your system, in respect to altering the storage X'40100'-X'40101' to X'FFFF'.

Supervisor Call Trace



For an explanation of:

- area_definition, see "ARea or JOBNAME Definition" on page 111.
- jobname, see "ARea or JOBNAME Definition" on page 111.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

svc

defines a certain Supervisor Call Code. You may define up to 16 different SVC codes. Specify the SVC code in hexadecimal notation.

If you specify more than one SVC code, the codes must be enclosed in parentheses and separated by one or more blanks, or by a comma with or without one or more blanks.

*****

(asterisk) defines that all SVC instructions are to be traced.

For a description of the trace and an example of the output, see "SVC Trace (Supervisor Call)" on page 67.

Statement Examples

```
TRACE SVC=* AR=BG ADD=0:*

TRACE SVC=A AR=BG

TRACE SVC=(1D 25) AR=BG
```

The statements shown:

- Trace all BG SVCs in BG partition and in system areas;
- Trace set timer SVCs (X'A') in BG partition;
- Trace BG partition WAITM and STXIT AB SVCs.

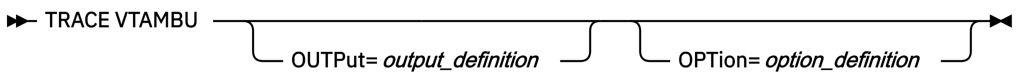
Initialization Examples

```
// EXEC SDAID
OUTDEV T=280
TRACE SVC=* AR=BG ADD=0:* -
            OPT=NOJCL
/*
```

The following items are covered by the trace setup shown:

- Use the SYSRDR device to set up the trace.
- Trace all supervisor call instructions.
- The SVC instructions from the BG partition are to be traced.
- Output the trace data to the tape on device address 280.
- Do not trace the job control branch instructions.

VTAM BUffer Trace



For an explanation of:

- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

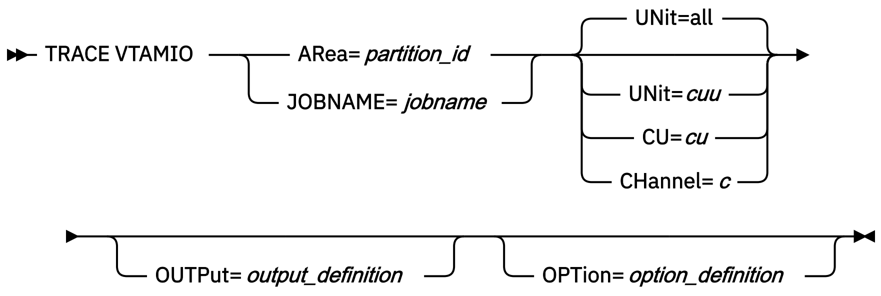
For a description of the trace and an example of the output, see "VTAMBU Trace (VTAM Buffer)" on page 67.

Statement Example

```
TRACE VTAMBU
```

The example defines a VTAM buffer trace.

VTAMIO Trace



For an explanation of:

- area_definition, see "ARea or JOBNAME Definition" on page 111.
- jobname, see "ARea or JOBNAME Definition" on page 111.
- cuu, cu and c, see "I/O Device Definition" on page 113.
- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

A VTAMIO trace requires an area definition. Define the operands as shown above.

For a description of the trace and an example of the output, see "VTAMIO Trace" on page 68.

Statement Example

```
TRACE VTAMIO AREA=F3
```

The example defines a VTAMIO trace for the F3 tasks.

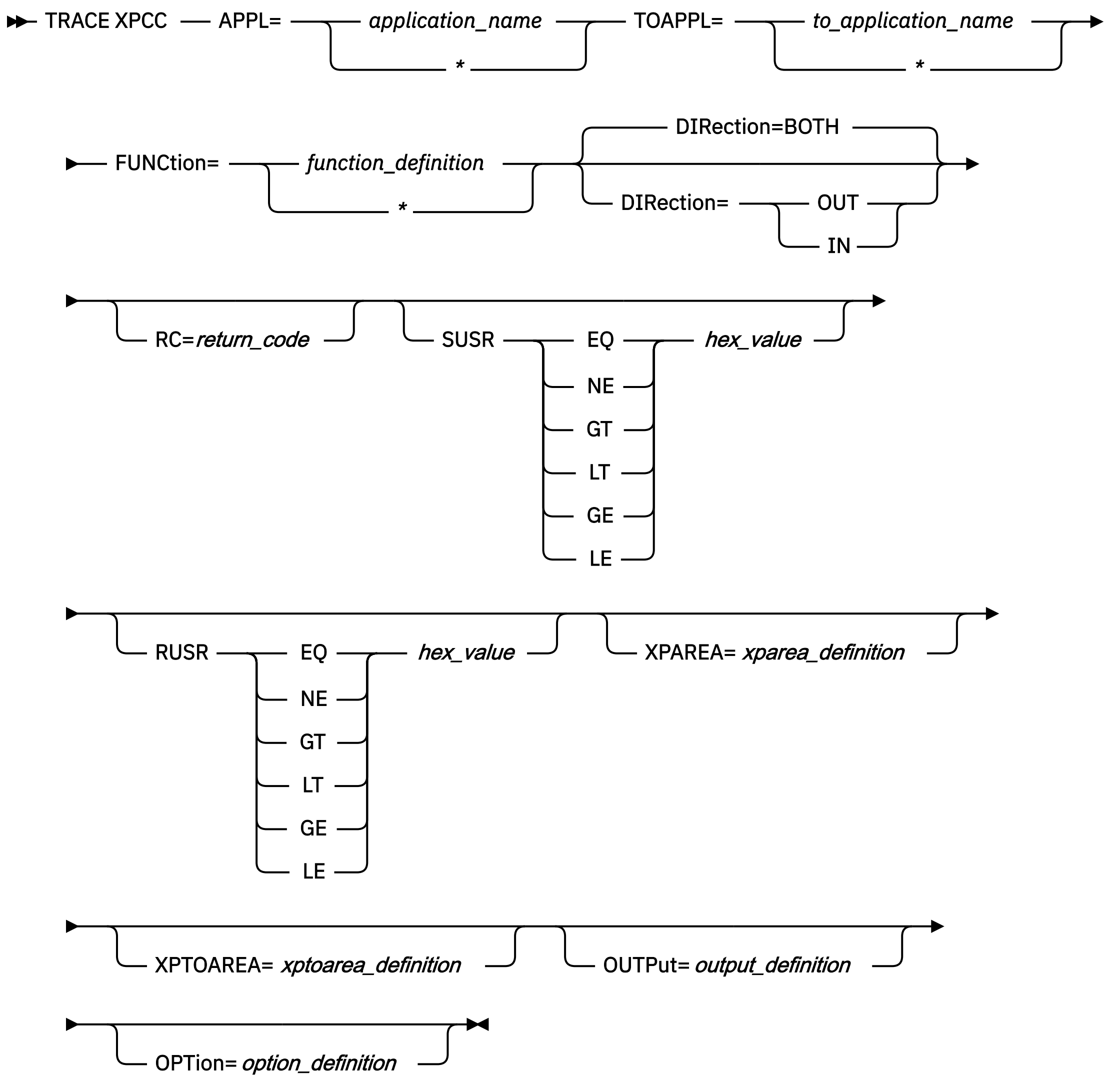
Initialization Example

```
// EXEC SDAID
OUTDEV T=280
TRACE VTAMIO AR=F3 -
             UN=020
/*
```

The following items are covered by the trace setup shown:

- Use the SYSRDR device to set up the trace.
- Trace all VTAM I/O operations concerning unit at address 020 and F3 tasks.
- Output the trace data to the tape on device address 280.

XPCC Trace



For an explanation of:

- output_definition, see "OUTPut Definition" on page 113.
- option_definition, see "OPTion Definition" on page 116.

APPL=*application_name* | *nnn | ***

An application name may consist of up to eight characters which must follow the naming conventions described in the XPCC invocation macro.

You can enter an application name as one of the following:

application_name

Only XPCC requests will be traced that have the specified name.

nnn*

If a character string is followed by an asterisk (*), all requests will be traced to those XPCC requests whose names begin with the character string.

*****

If you specify an asterisk (*), all requests will be traced for all XPCC requests.

TOAPPL=*to_application_name* | *nnn | ***

The rules for using TOAPPL are the same as for APPL (above).

FUNCtion=TERMQsce | TERMPrg | TERMIn | DISCAll | DISCPrg | DISCOnn | SENDI | SENDR | SEND | RECeive | REPly | IDent | COnnect | CLear | PUrge | *

FUNCtion can take any of the functions listed above, or asterisk (*). If you enter one of the functions, only XPCC events of the specified function will be traced. If you enter an asterisk (*), XPCC events for *all* functions will be traced.

DIRection=IN | OUT | BOth

You can enter a DIRection as one of the following:

DIR=IN

Only incoming XPCC events will be traced.

DIR=OUT

Only outgoing XPCC events will be traced.

DIR=BOth

Both incoming and outgoing XPCC events will be traced.

If this parameter is omitted, DIR=BOth is used as default.

RC=*return_code*

If you specify a value for *return_code*, only events with a final return code equal to this value will be traced. If this parameter is omitted, events with any final return code will be traced.

SUSR | RUSR *comparator hex_value*

Compares a specified string of hexadecimal characters with the contents of the corresponding field contained within the XPCC control block.

- SUSR corresponds to field IJBXSUSR in the XPCC control block.
- RUSR corresponds to field IJBXRUSR in the XPCC control block.
- *comparator* can be one of: EQ, NE, GT, LT, GE, or LE.
- *hex_value* is a string of up to sixteen hexadecimal characters. Characters within this string that should *not* be compared can be substituted by a dot (**.**).

Here are some examples of the use of SUSR:

```
SUSR EQ FFFFFFFFFFFFFFFF  (check all hex characters in IJBXSUSR)
SUSR NE 1020              (check only first four hex characters)
SUSR LT ......FFD         (check only characters 7 to 9)
SUSR GT FF...FF..         (THIS IS AN INVALID STATEMENT!)
```

If the SUSR | RUSR parameter is omitted, no checking will be performed.

XPAREA=*syslog_id*

Defines the partition where the application that is defined using APPL=*application_name* is running, in a pair of two interacting partitions. If the parameter is omitted, all XPAREAS are assumed.

XPTOAREA=*syslog_id*

Defines the partition where the application that is defined using TOAPPL=*application_name* is running, in a pair of two interacting partitions. If the parameter is omitted, all XPTOAREAS are assumed.

Note: Using the OUTPut=*output_definition* parameter (described in "OUTPut Definition" on page 113), you can request this type of output:

```
OUTPUT=XPCCB     (prints the XPCC Control Block)
OUTPUT=XPDATABU  (prints the contents of the Transmit Data Buffer)
```

For a description of the trace and an example of the output, see "XPCC Trace" on page 69.

Statement Examples

```
TRACE XPCC APPL=* TOAPPL=* FUNC=* OUTPUT=XPCCB

TRACE XPCC APPL=RESI TOAPPL=* FUNC=SEND DIR=OUT XPAREA=BG
XPTOAREA=F7 OUTPUT=XPDATABU
```

In the above two examples, the TRACE statements shown:

1. Trace all XPCC events.
2. Trace outgoing XPCC SEND events from application RESI to any partner application, but only if RESI runs in BG and the partner application is in F7.

Initialization Example

```
// EXEC SDAID
OUTDEV T=280
TRACE XPCC APPL=* TOAPPL=* FUNC=*
/*
```

The following items are covered by the trace setup shown:

- Use the SYSRDR device to set up the trace.
- Trace all XPCC events.
- Output the trace data to the tape on device address 280.

Additional Definitions

```
 ARea=, JOBNAME=, ADDRess=, OFFset=, PHase=, OPTion=, OUTPut=,
 UNit=, CHannel=, CU=
```

The following section describes definitions which may follow the trace type specification in the TRACE statement.

Table 9 on page 110 shows a list of all additional definitions, a summary of their function, and a reference to their format description and examples in this topic. The various definitions are described in detail under:

- "Defining the Area to be Traced: AREA Definition" on page 69.
- "Defining the Job to be Traced: JOBNAME Definition" on page 70.
- "Defining the Storage to be Traced: OFFset, ADDress, PHase, LTA" on page 70.
- "Defining Additional Trace Output: OUTPut Definition" on page 72.
- "Defining the Trace Options: OPTion Definition" on page 80.
- "Defining the Traced I/O Devices" on page 82.

Table 9. Additional Definitions Summary

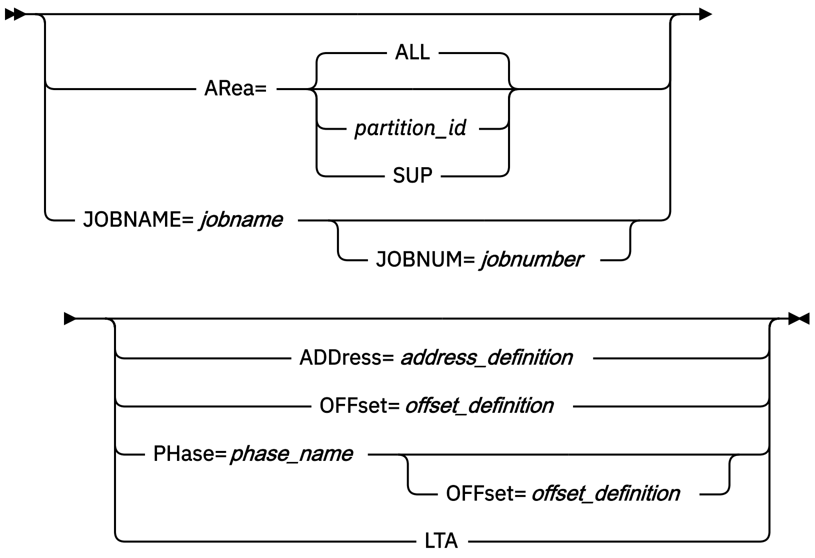
Operand	Function	See:
ARea	Limit tracing to a certain system area	"ARea or JOBNAME Definition" on page 111
JOBNAME	Limit tracing to a certain VSE/ POWER job	"ARea or JOBNAME Definition" on page 111
ADDress	Limit tracing to a certain address range	"ADDress Definition" on page 111
OFFset	Limit tracing in a partition or phase area	"OFFset Definition" on page 112
PHase	Limit tracing to a certain phase	"PHase Definition" on page 112
UNit	Define the device address	"I/O Device Definition" on page 113
CHannel	Define the channel address	"I/O Device Definition" on page 113
CU	Define the control unit address	"I/O Device Definition" on page 113

Table 9. Additional Definitions Summary (continued)

Operand	Function	See:
OUTPut	Define additional trace output	"OUTPut Definition" on page 113
OPTion	Define additional trace options	"OPTion Definition" on page 116

ARea or JOBNAME Definition

The possible storage area definitions together with ARea or JOBNAME are:


ARea=
ALL
partition_id
SUP
JOBNAME= jobname
JOBNUM= jobnumber
ADDress= address_definition
OFFset= offset_definition
PHase= phase_name
OFFset= offset_definition
LTA

ADDress=
See "ADDress Definition" on page 111.

OFFset=
See "OFFset Definition" on page 112 (not for ARea=ALL).

PHase=
See "PHase Definition" on page 112.

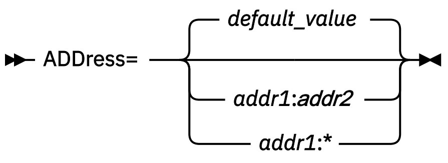
LTA
Defines the Logical Transient Area as tracing range.

Default Value

If you use ARea=partition-id|SUP or JOBNAME without an additional specification, OFFset=0:* is assumed. OFFset=0:* defines the whole partition (or the area between zero and end-of-supervisor) as trace area.

If you use ARea=ALL without an additional specification, ADDR=0:* is assumed (that is, all virtual storage).

ADDress Definition



You can limit the trace to a certain address range within the storage allocated to VSE with the ADDress definition.

addr1:addr2

Defines a trace address range in hexadecimal notation in any virtual storage defined to VSE.

For example:

```
ADD=500000:*
```

Default Value

If you omit the ADDress specification, the default trace address range depends upon the specifications in the ARea= or JOBNAME= parameters.

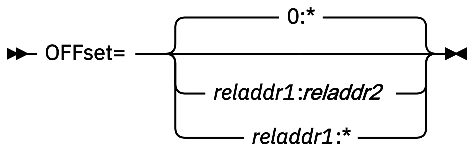
If you use ARea=*partition_id*, the complete storage allocated to the partition is assumed to be trace address range.

If you use JOBNAME=, the complete storage allocated to the specified JOB is assumed to be trace address range.

If you use ARea=ALL, then ADDR=0:* is assumed (that is, the complete storage range is allocated to VSE).

Note: If you use ARea=*partition_id* and parts of your program are located in the SVA, you must specify ADDR=0:* to include these parts of your program in the trace.

OFFset Definition



You can limit the trace to a certain address range via offsets relative to the defined partition, supervisor or phase with the OFFset definition.

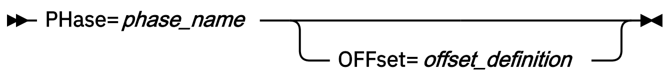
For example:

```
OFF=200:*
```

Default Value

If you omit the OFFset definition, 0:* is assumed.

PHase Definition



With the PHase definition the traced storage area is defined by the area occupied by that phase.

phase-name

For example:

```
PH=PROGRAM1
```

Default Value

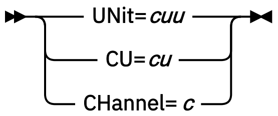
You may limit the traced storage area within the defined phase with the OFFset keyword operand. See "OFFset Definition" on page 112.

For example:

```
PH=PROG OFF=20:400
```

The example above initializes a trace which is active only in the phase with the name PROG between program address X'20' to X'400'.

I/O Device Definition



UNit=cuu

For example:

```
UN=280
UN=(280 310)    Use the parentheses if you
                specify more than one address.
UN=e            Same as 00e.
```

CU=cu

For example:

```
CU=28
CU=00
CU=(28 31)      Use the parentheses if you
                specify more than one address.
```

CHannel=c

For example:

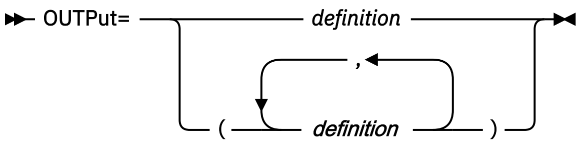
```
CH=2
CH=(2 3)        Use the parentheses if you
                specify more than one address.
```

Only one of the parameters UNit, CHannel, or CU can be specified in the same TRACE command.

Default Value

If none of the I/O parameters is specified, all devices are traced.

OUTPut Definition



You may take one or more definitions together with one TRACE statement. If you enter more than one OUTPut definition in direct input mode, enclose them in parentheses.

A summary of all definitions which you can specify with OUTPut= is given in Table 10 on page 113. This table contains the format and a short description of the data which is recorded together with the trace event record. For those output definitions which allow additional definitions, a reference to the information contained in this topic is shown.

Table 10. OUTPut Definition Summary

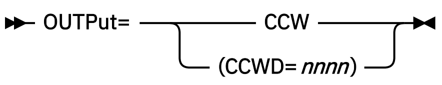
Definition	What it records/prints in addition:	See:
BUffer	Contents of SDAID output buffer	-

Table 10. OUTPut Definition Summary (continued)

Definition	**What it records/prints in addition:**	**See:**
CCB	CCB or IORB (TRACE=IO, SSCH, or VTAMIO only)	-
CCW	CCWs, IRB (TRACE=IO, SSCH, or VTAMIO only)	"Recording CCW" on page 114
CCWD=nnnn	CCWs plus nnnn bytes of data, IRB (TRACE=IO, SSCH, or VTAMIO only)	"Recording CCW" on page 114
COMReg	Partition communication region	-
CReg	Control registers	-
DUMP	Virtual storage	"Dumping Virtual Storage" on page 115
FReg	Floating point registers	-
GReg	General purpose and access registers	-
IOTab	PUB, LUB, ERBLOC, ERRQ, CHANQ	-
LOCKTE	Lock table entry (LOCK trace only)	"LOCK / UNLOCK Trace" on page 60
LOwcore	Processor storage from zero to X'2FF'	-
LTA	Logical transient area	-
PTA	Physical transient area	-
PTAB	Partition related control blocks: PCB, PIB, PIB2	-
SUPvr	Supervisor plus GREG and CREG	-
SYSCom	System communication region	-
TOD	Time-of-Day clock	-
TTAB	Task related control blocks: TIB, TCB, PCB, PIB, PIB2	-
XPCCB	XPCC control block (XPCC trace only)	"XPCC Trace" on page 69
XPDATABU	Buffer for data to be transmitted	"XPCC Trace" on page 69

Note: A description of all output definitions is given under "Defining Additional Trace Output: OUTPut Definition" on page 72.

Recording CCW



CCW

(channel command word) records/prints the available channel program (CCW chain) plus the CCB and the TOD clock when the trace type is SSCH.

In case of an IO trace only the CCWs which refer to transferred data are recorded or printed.

Specifying this output option for an event other than IO or SSCH is not meaningful.

CCWD=nnnn

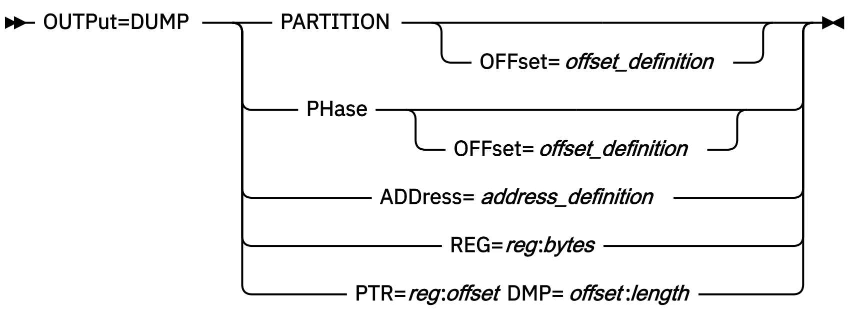
(CCW plus data) records/prints up to a maximum of nnnn bytes of the transferred data, the CCB and the TOD clock in addition to the information processed with the CCW specification. The number nnnn may be any (decimal) number between 1 and 65535.

The most meaningful trace type to be combined with this output option is the IO trace.

For an example of the output produced with this option, see Figure 39 on page 74.

You may define either CCW or CCWD=nnnn.

Dumping Virtual Storage


OUTPut=DUMP
PARTITION
OFFset= offset_definition
PHase
OFFset= offset_definition
ADDress= address_definition
REG=reg:bytes
PTR=reg:offset DMP= offset:length

DUMP

records or prints the contents of virtual storage.

You may request up to ten different dumps.

You have to specify one or more of the dump area specifications as shown below.

PARTITION

For example, dump the storage beginning with offset X'0' up to X'78' of the partition for which the trace is active.

```
OUTP=(DUMP PARTITION OFF=0:78)
```

PHase

For example, dump the area starting with relative address X'40' up to relative address X'60' in the phase defined via the ‘PHase=’ keyword operand.

```
OUTP=(DUMP PH OFF=40:60)
```

ADDress

For example, dump the contents of two bytes starting on storage location 0080 (hexadecimal). The definition in direct input mode looks like this:

```
OUTP=(DUMP ADD=80:81)
```

REG=reg:bytes

For example, dump 16 bytes of storage pointed to by register 15.

```
OUTP=(DUMP REG=F:10)
```

PTR=reg:offset DMP=offset:length

For example, dump a four-byte field which is located in a table with an offset of X'20' bytes. The table address is stored in storage pointed to by register 6 plus displacement X'100':

```
OUTP=(DUMP PTR=6:100 DMP=20:4)
```

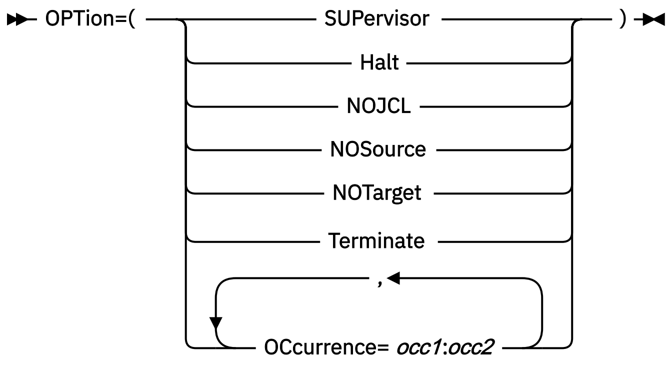
Trace Statement Example: Dump an Area in a Phase

```
TRACE PGMC=* -
      AR=BG -
      PH=PHASE1 -
      OUTP=(GREG DUMP PH -
           OFF=0:400 LOWC)
```

The following items are covered by the trace setup shown in the example:

- Trace program check interrupts
- Traced tasks: BG partition main and subtasks
- Traced storage area: phase1 storage area
- Additional trace output:
  - general registers (GREG)
  - dump of X'400' bytes of phase1 area starting at relative address 0 (DUMP PH OFF=0:400)
  - low-core (LOWC).

OPTion Definition



For a description of the OPTion definitions, see "Defining the Trace Options: OPTion Definition" on page 80.

OCCurrence Examples

```
OPT=OCC=1:1          Trace only the first occurrence
                     of the event

OPT=OCC=1:*          Trace all occurrences of the
                     event (this is the default value)

OPT=OCCUR=5:12       Trace selected occurrences
                     (5 to 12) of the specified event
```

Chapter 10. Initialize an SDAID Trace via a Procedure

This topic describes how you initialize SDAID traces by using just one job control (JCL) EXEC PROC statement. VSE/Advanced Functions offers a set of predefined JCL procedures to initialize SDAID traces under control of a partition. These procedures are included in the system sublibrary IJSYSRS.SYSLIB.

The most frequently used SDAID functions are covered by these JCL procedures. The JCL procedures contain reasonable default values to ease the SDAID trace initialization process. You may define your own procedures tailored to the requirements of your installation or to a special debugging problem.

Introduction

Besides the direct input mode and prompt mode trace initialization a third initialization method is available under VSE, the initialization via cataloged procedures. These cataloged procedures contain direct input mode command skeletons. You activate the initialization via the job control EXEC PROC statement.

```
// EXEC PROC=trace-type,specification,specification,...
```

The specifications in the EXEC PROC statement are translated to SDAID direct input mode command operands.

Each EXEC PROC statement contains the name of the procedure (trace-type) plus additional specifications. You may define the specifications in any order. A continuation sign has to follow the comma if you use the console for input. If you use SYSIN to enter the procedure statement, the continuation sign has to be in column 72 and the continuation line must start in column 16.

Notational Conventions

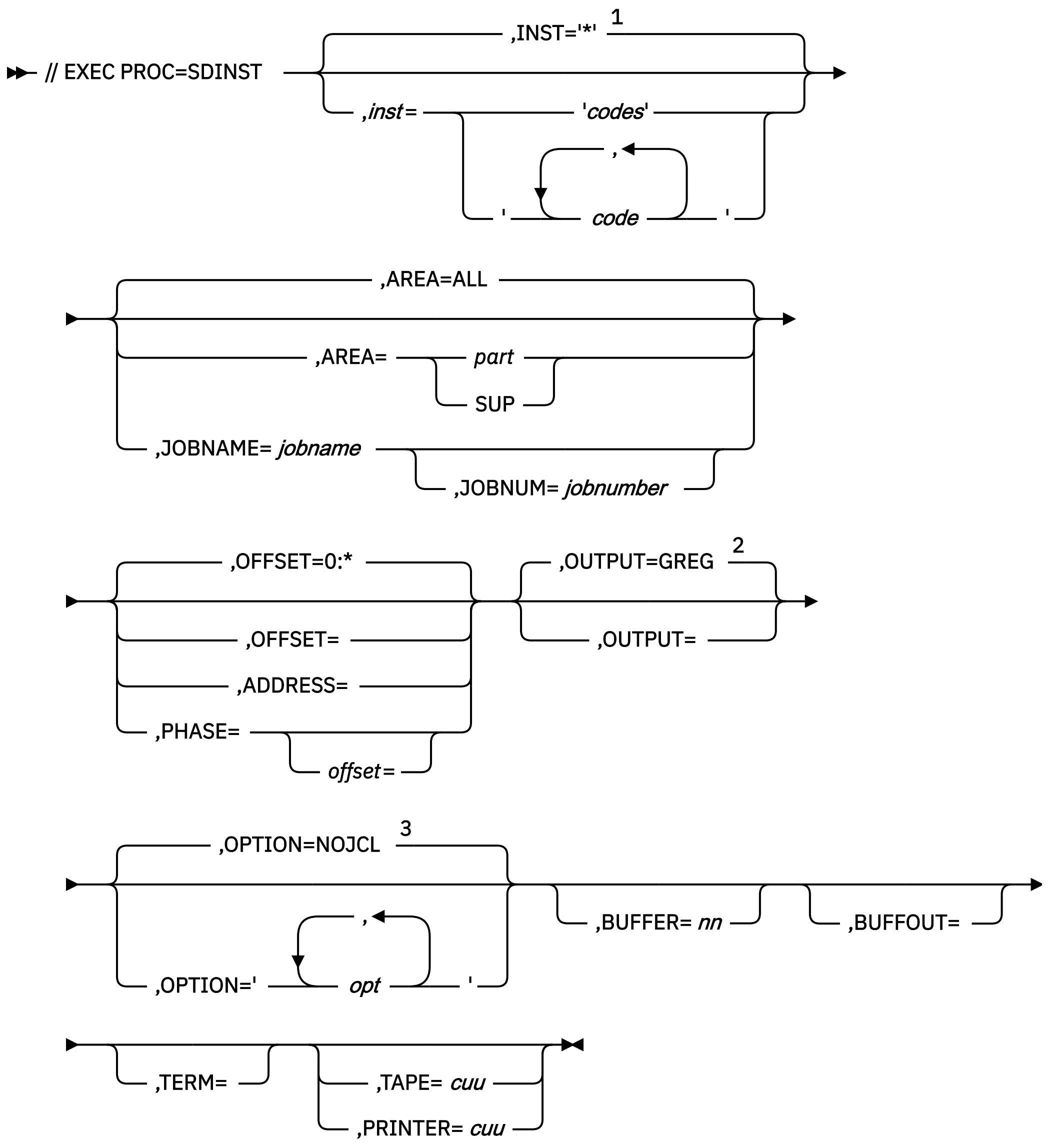
In this topic, the EXEC PROC statements needed to initialize the various SDAID traces are described in detail. The notation of each EXEC PROC statement description shows you:

- Which operands are optional, and which are mandatory;
- What the default values of the operands are.

The syntax of the EXEC PROC statement follows the conventions for job control statements as described in *z/VSE System Control Statements*.

The SDAID command defaults are not shown in this topic. The defaults for each trace type are given in "Summary of TRACE Types" on page 55.

Figure 49 on page 118 shows a sample description of the EXEC PROC statement used to call a trace. The handling of the operands and their default values is described in the section following the figure.



Notes:

[1] The default values are defined in the cataloged procedure SDINST.
[2] The default values are defined in the cataloged procedure SDINST.
[3] The default values are defined in the cataloged procedure SDINST.

Figure 49. SDINST Sample Procedure

Default Value Considerations

In an EXEC PROC statement, you can specify the SDAID trace operands in three ways:

1. Using the default value defined in the procedure;
2. Using the default value of the SDAID trace command itself;
3. By specifying a value of your choice in the EXEC PROC statement.

To use a **default defined in the procedure**, simply omit the appropriate operand from the EXEC PROC statement. If the procedure has no default for the operand, this will cause the SDAID default to be used.

To use the **SDAID default value**, nullify the operand in the procedure by coding ‘keyword=’ in the EXEC PROC statement. For example:

```
EXEC PROC=SDINST,OUTPUT=
```

would cause the SDAID default value for OUTPUT in the trace command to be used.

An operand which has no procedure-defined default value does not have to be nullified. Simply omit the operand from the EXEC PROC statement.

To **specify a value of your choice**, include the appropriate keyword and value in the EXEC PROC statement. This overrides the procedure-defined default, if any, and the SDAID default value. For example:

```
EXEC PROC=SDINST,AREA=BG,OUTPUT=PTAB
```

overrides the procedure defined OUTPUT value GREG. The trace runs as if OUTPUT=PTAB had been specified in the SDAID trace command.

Writing Cataloged Procedures

You can create and catalog your own procedures for particular problem-determination situations.

For example, you can define additional default values, or you can create a procedure for a trace type for which no procedure has been cataloged.

When you write a procedure, consider that you have to follow the correct command-input sequence. For example, the TRACE= definition for some trace types has to be followed by the AREA or JOBNAME specification. You can use the figures shown under "Command Input Path Example" on page 136 to establish the correct command-input sequence.

The Statements of a Cataloged Procedure

The result of the execution of each EXEC PROC statement is a complete direct-input trace initialization. The direct input mode statements are cataloged as:

- Fixed definitions;
- Placeholder definitions;
- Placeholder definitions with default values.

Fixed Definitions

are those definitions in the cataloged procedure which are always active. They cannot be altered or overridden by values specified in the EXEC PROC statement. Code them as you would in direct-mode trace initialization.

Placeholder Definitions

can be replaced by a value which you specify in the EXEC PROC statement. These are handled as follows:

- A placeholder, beginning with an ampersand (&), takes the place of the value after the equals sign (=) in the cataloged trace command (for example: UNIT=&UNIT);
- The placeholder name, without the ampersand, is used in the EXEC PROC statement to provide a definition at execution time (for example: EXEC PROC=SDIO,**UNIT=280**).

The statement in the cataloged procedure:

```
UNIT=&UNIT
```

Your definition in the EXEC PROC statement:

```
UNIT=280
```

The created direct-input-mode statement:

```
UNIT=280
```

If the operand of a trace statement can have a list of values after the equals sign, one placeholder is still enough. In the EXEC PROC statement, the list must be enclosed in single quotes (this is a requirement of job control). SDAID replaces these quotes with parentheses in the trace initialization statement which the procedures produces. For example:

The statement in the cataloged procedure:

```
UNIT=&UNIT
```

Your definition in the EXEC PROC statement:

```
UNIT='280 281'
```

The created direct-input-mode statement:

```
UNIT=(280 281)
```

Placeholder with Default Value Definitions

You can define default values for placeholders in cataloged procedures. The default value must follow the placeholder and be enclosed in "less-than" (<) and "greater-than" (>) signs (for example: UNIT=&UNIT<280>). In the IO trace procedure in Figure 50 on page 120, the default output value is specified as follows:

```
OUTPUT=&output<CCWD=256>
```

If you omit OUTPUT=value definition from the EXEC PROC statement, the default value is inserted in the direct input statement, which is generated as:

```
OUTPUT=CCWD=256
```

If you do not want to provide any definition, and also want to avoid the procedure default, you must code:

```
OUTPUT=
```

(with no value) in the EXEC PROC statement. The operand in the procedure is nullified. No OUTPUT definition is inserted in the created direct input statement.

Figure 50 on page 120 shows an example of a fixed definition, a placeholder definition, and a placeholder with a default value definition. The cataloged procedure in this example is called by the member name under which you cataloged it.

```
// EXEC SDAID
TRACE SSCH AREA=&area              -
           JOBNAME=&jobname        -
           UNIT=&unit              -
           OUTPUT=TOD              -     Fixed Definition
           OPTION=&option
TRACE IO   AREA=&area              -     Placeholder Definition
           JOBNAME=&jobname        -
           UNIT=&unit              -
           OUTPUT=&output<CCWD=256> -     Placeholder with Default
                                         Value Definition
           OPTION=&option
```

Figure 50. Example: Cataloged Procedure

In the two trace types which are initialized:

- The same AREA, UNIT and OPTION values are used for both traces. These values are specified in the EXEC PROC statement;
- If you do not specify the OUTPUT operand, the default OUTPUT=CCWD=256 is defined for the IO trace;
- The SSCH trace event record always contains the time-of-day clock.

Procedures to Initialize SDAID Traces

This section describes the trace procedures available with VSE/Advanced Functions to initialize SDAID traces. The additional keyword operands which you find in the trace procedure statements are described under "Additional Keyword Operands in Trace Procedure Statements" on page 129.

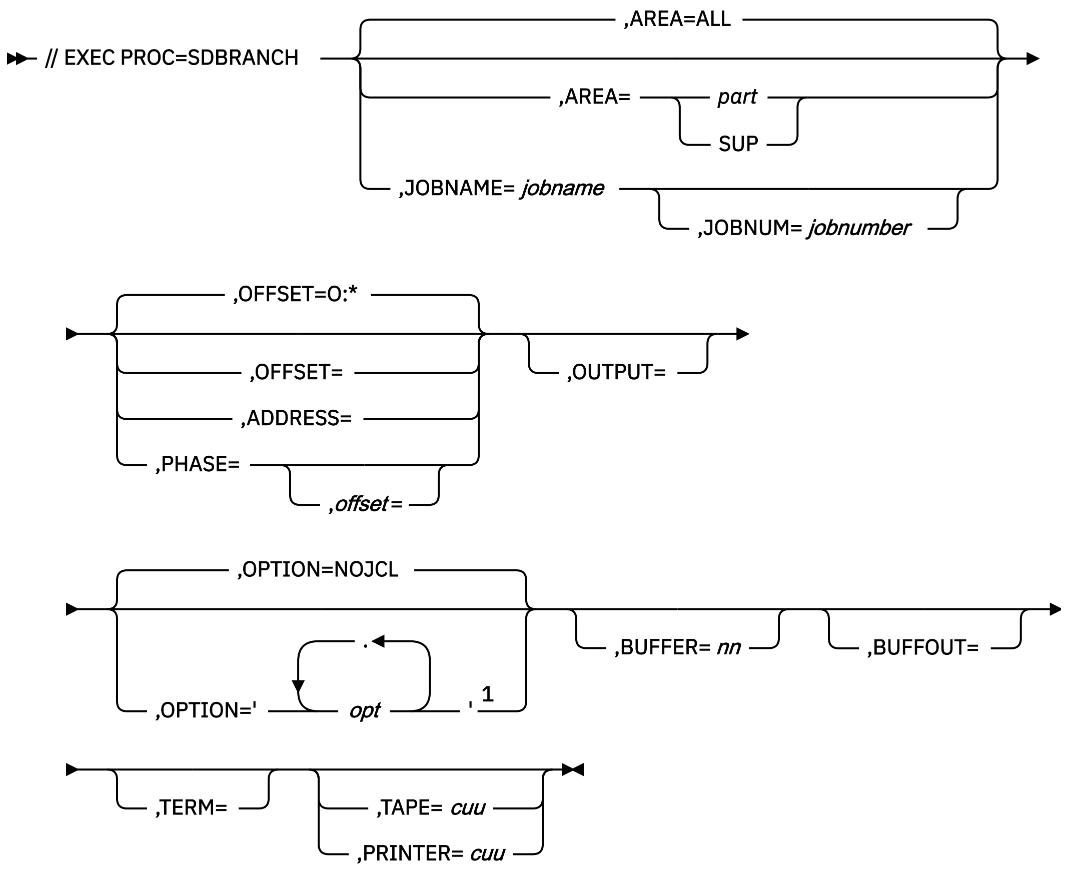
Choose the appropriate procedure from Table 11 on page 121.

Summary of Trace Procedures

Table 11. Trace Procedures Summary

Procedure	Provides Information on:	See:
SDBRANCH	Successfully executed branch instructions	"Branch Trace Initialization" on page 121
SDINST	Selected or all instruction(s) execution	"Instruction Trace" on page 123
SDIO	I/O interrupts and SSCH instructions	"SSCH and I/O Interrupt Trace" on page 124
SDLOAD	Phase load requests, or actual load	"Fetch/Load Trace" on page 125
SDPGMC	Program check interruptions	"Program Check Trace" on page 126
SDSTOR	Storage alterations	"Storage Alteration Trace" on page 127
SDSVC	Executed supervisor calls	"SVC Trace" on page 128

Branch Trace Initialization



Notes:

[1] Up to 6 options may be specified.

See the “Additional Keyword Operands in Trace Procedure Statements” on page 129.

The procedure SDBRANCH initializes traces for all branch instructions which actually caused a branch.

Find the description of the trace type and an example of the output under “BRANCH Trace” on page 56.

Defaults Set in the Procedure

OPTION=NOJCL is active if you omit OPTION=.

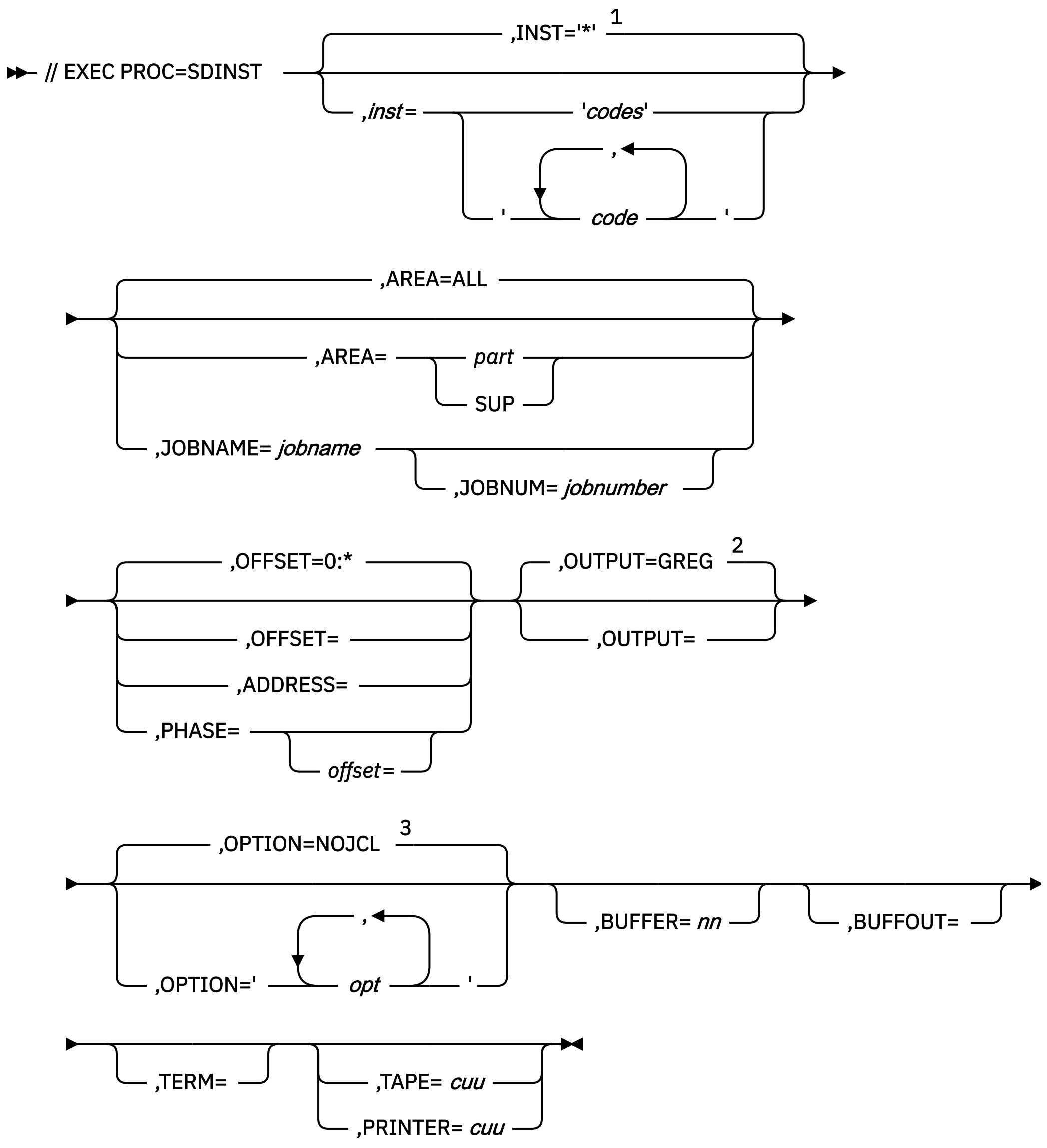
Statement Example

Here are the items of the trace setup shown below:

- Trace type: BRANCH
- Area for which events are collected: storage address 80010 up to address 80100
- Traced tasks: F4 main task and its subtasks
- Output destination: Tape with device address 280
- Avoid the tracing of JCL instructions (default)

```
// EXEC PROC=SDBRANCH,AREA=F4,ADDRESS='80010:80100',TAPE=280
```

Instruction Trace



Notes:

[1] The default values are defined in the cataloged procedure SDINST.
[2] The default values are defined in the cataloged procedure SDINST.
[3] The default values are defined in the cataloged procedure SDINST.

See the "Additional Keyword Operands in Trace Procedure Statements" on page 129.

The procedure SDINST initializes traces for all instructions or for selected instructions executed within a specified area. Find the description of the trace type and an example of the output under "INSTRUCTION Trace" on page 59.

Defaults Set in the Procedure

If you omit INST, all instructions are traced (INST='*' is the default). OPTION=NOJCL and OUTPUT=GREG are assumed if you omit both these operands.

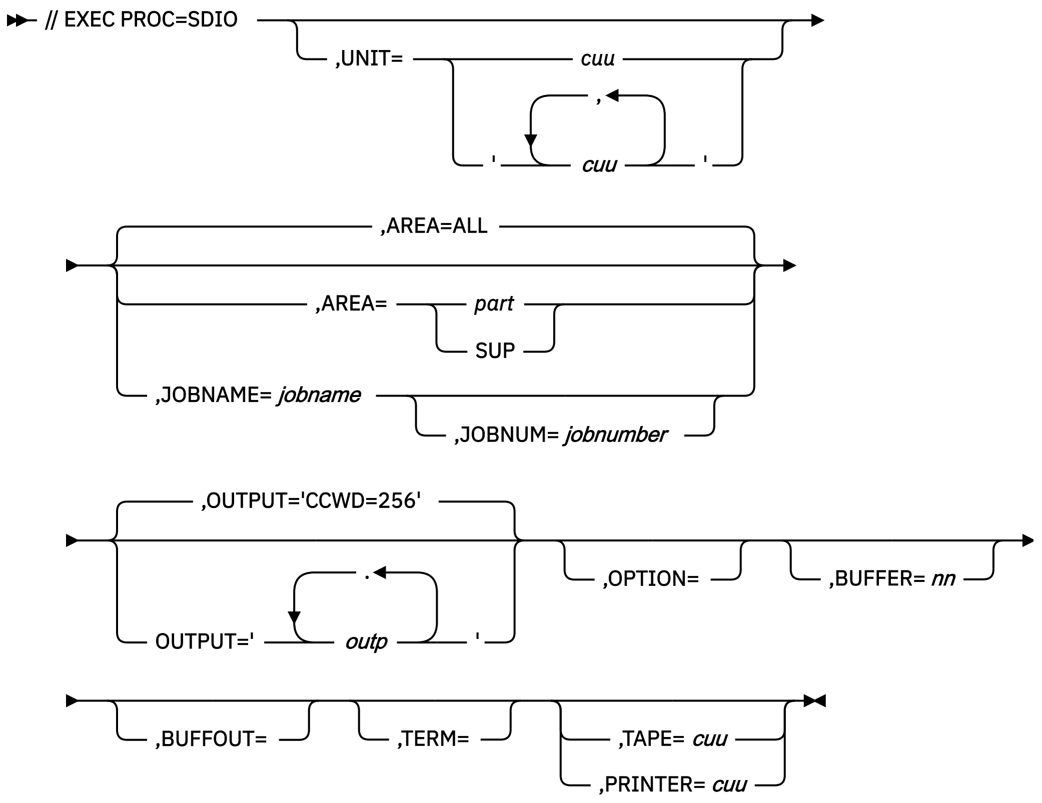
Statement Example

Here are the items of the trace setup shown below:

- Trace type: INSTRUCTION
- Trace all instructions (default)
- Area for which events are collected: storage address 40328 up to address 40350
- Traced tasks: BG main and subtasks
- Additional Output: default GREG output
- Output destination: 16K bytes buffer
- Output device for buffer: tape with device address 281
- Event to write the buffer to tape: program check in BG partition
- Avoid the tracing of JCL instructions (default)
- Note that the continuation sign has to follow the comma if you use the console for input. If you use SYSIN to enter the procedure statement the continuation sign has to be in column 72.

```
// EXEC PROC=SDINST,AREA=BG,ADDRESS='40328:40350',-
                             BUFFER=16,BUFFOUT=PGMC,T=281
```

SSCH and I/O Interrupt Trace



See the "Additional Keyword Operands in Trace Procedure Statements" on page 129.

The procedure SDIO initializes the SSCH instructions and I/O interruptions trace.

Note that the TOD clock entry is added to each SSCH instruction event record.

Find the description of the trace types and examples of the output under "IO Trace (I/O Interrupt)" on page 59 and "SSCH Instruction Trace" on page 65.

Default Set in the Procedure

If you do not define UNIT, all devices are traced.

If you omit the AREA definition, all tasks in the system are traced.

These are both SDAID defaults.

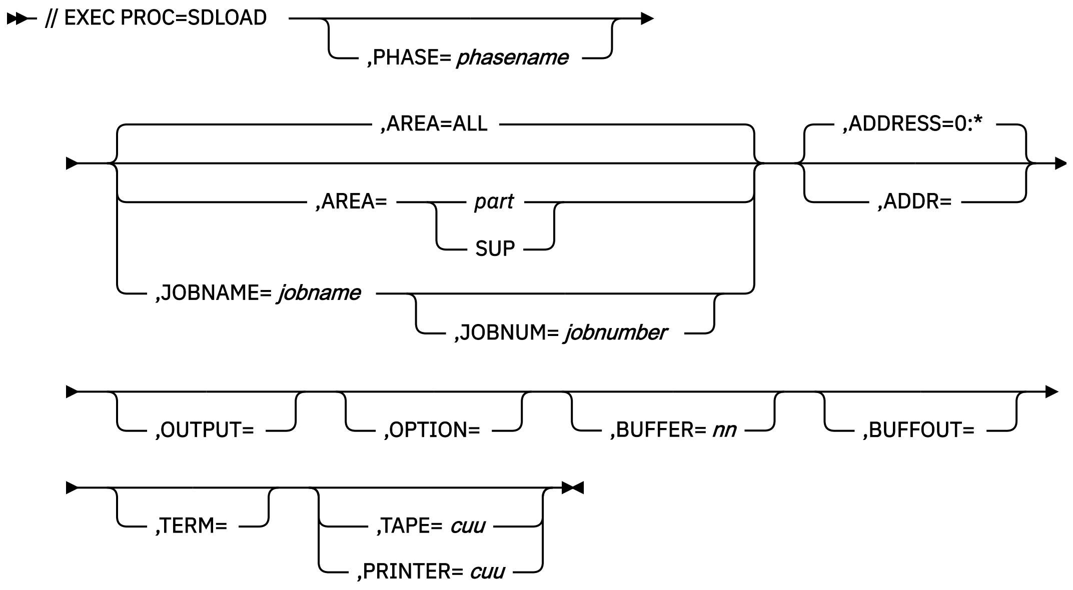
OUTPUT='CCWD=256' is the default definition for the I/O interrupt trace.

Statement Example

- Trace types: IO, SSCH
- Traced tasks: SDAID default value used (ALL)
- Traced unit: 281
- Additional Output: procedure default CCWD=256
- Output destination: printer with device address 00E

```
// EXEC PROC=SDIO,UNIT=281,PRINTER=00E
```

Fetch/Load Trace



See the “Additional Keyword Operands in Trace Procedure Statements” on page 129. The procedure SDLOAD initializes traces for all phase load requests and phase load operations.

For the description of the trace type and an example of the output, see “PGMLOAD (Fetch/Load) Trace” on page 64.

Defaults Set in the Procedure

ADDRESS='0:*' is defined if you omit ADDRESS=.

Statement Example

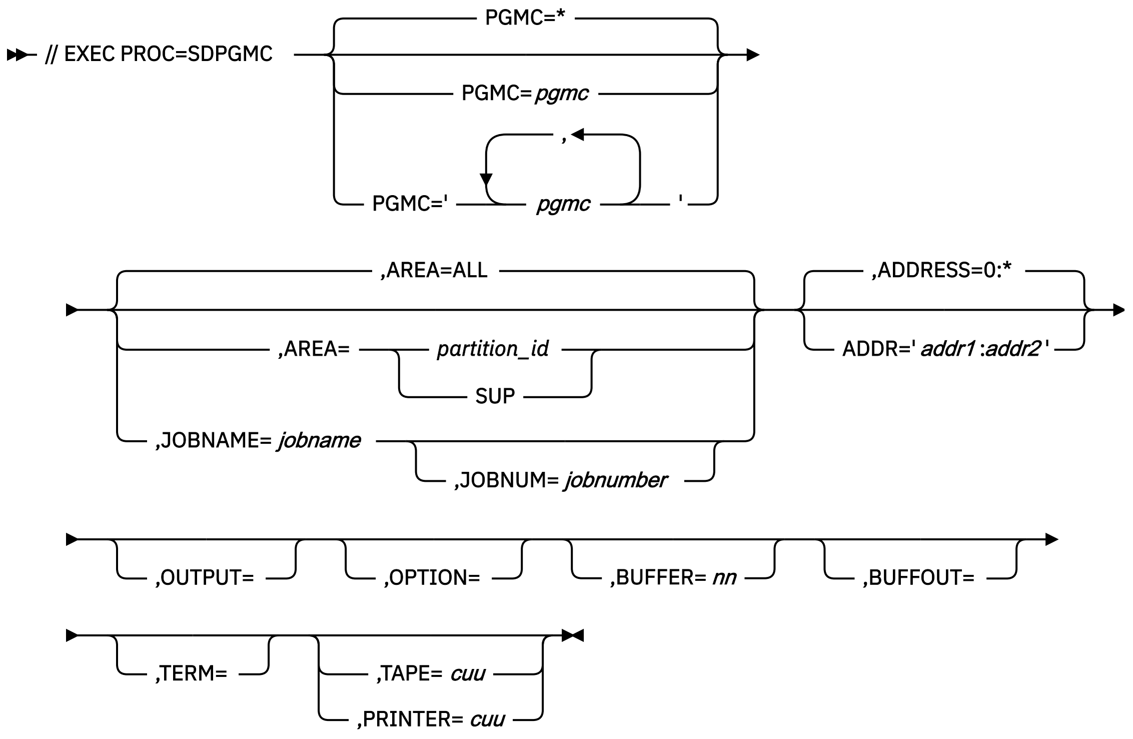
Here are the items of the trace setup shown below:

- Trace type: PGMLOAD
- Traced tasks: all tasks of the BG partition
- Traced storage area: whole VSE/Advanced Functions storage (default)
- Phase whose fetch/load operation is to be traced: MYPHASE
- Additional Output: dump of the storage contents with the address 0 to X'3000', relative to the BG partition start address on occurrence of the PGMLOAD trace event.
- Output destination: tape with device address 280

- Note that the continuation sign has to follow the comma if you use the console for input. If you use SYSIN to enter the procedure statement the continuation sign has to be in column 72.

```
// EXEC PROC=SDLOAD,PHASE=MYPHASE,AREA=BG,OUTPUT='DUMP PART OFF=0:3000',-
                    TAPE=280
```

Program Check Trace



See the “Additional Keyword Operands in Trace Procedure Statements” on page 129.

The procedure SDPGMC initializes traces for program check interruptions.

For a description of the trace type and an example of the output, see “PGMCheck Trace (Program Check)” on page 63.

Defaults Set in the Procedure

ADDRESS='0:*' is defined if you omit ADDRESS=.

All program check interrupts are traced if you omit PGMC=.

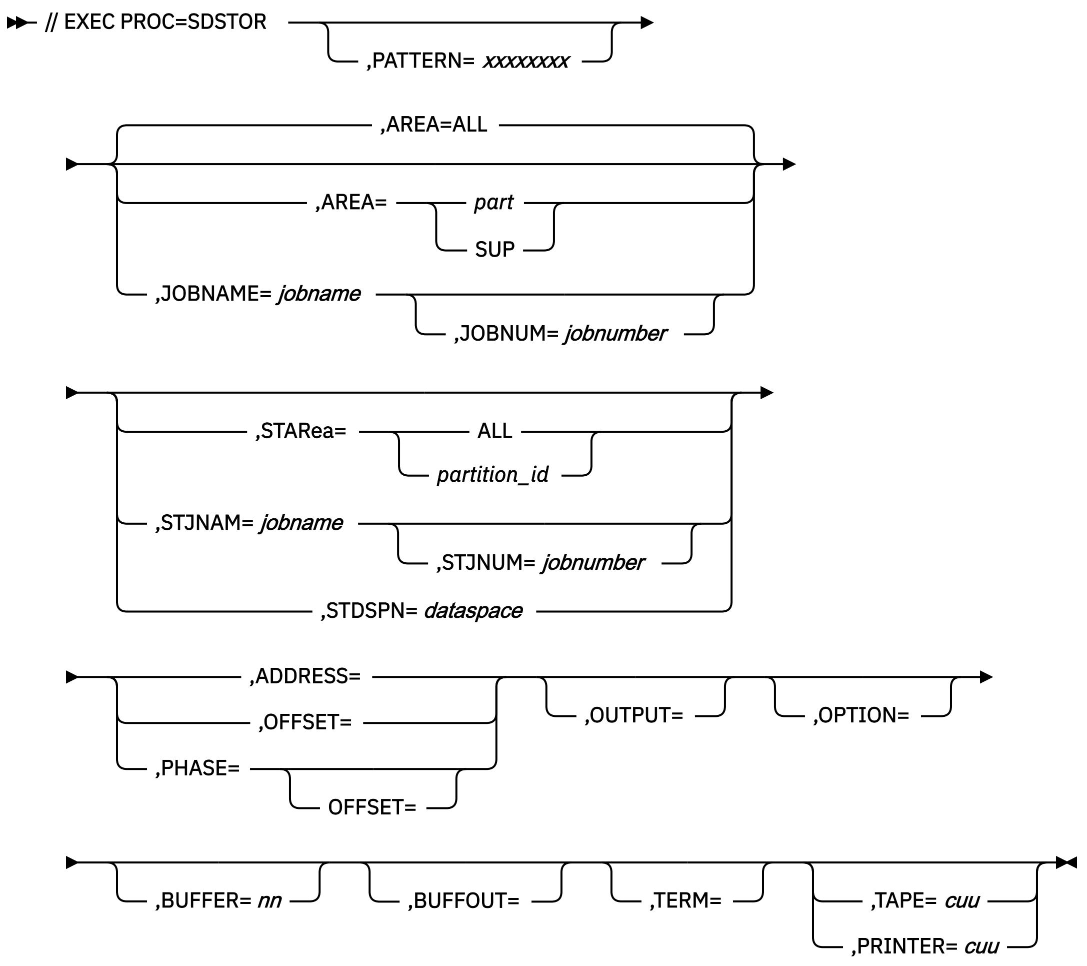
Statement Example

Here are the items of the trace setup shown below:

- Trace type: PGMCHECK
- Traced tasks: all tasks of the BG partition
- Traced storage area: BG partition area (OFF=0:* defined by SDAID defaults)
- Additional Output: dump of the storage contents with the address 0 to X'5000', relative to the BG partition start address on occurrence of the PGMCHECK trace event.
- Output destination: tape with device address 280

```
// EXEC PROC=SDPGMC,AREA=BG,OUTPUT='DUMP PART OFFSET=0:5000',TAPE=280
```

Storage Alteration Trace



See the "Additional Keyword Operands in Trace Procedure Statements" on page 129.

The procedure SDSTOR initializes traces for storage alterations.

You use this trace type as a tool to find those instructions which modify a certain storage area. In most cases you do not know which phase in your system alters this area. For this, define AREA=ALL to watch all tasks operating in your system. The observed storage area is defined via the ADDRESS= keyword.

The optional keyword 'PATTERN=' restricts monitoring to those instructions which change the storage contents into the defined pattern. The specified storage interval which you define with the ADDRESS= keyword should have the same length as the specified pattern (if any).

For a description of the trace type and an example of the output, see "STORAGE Alteration Trace" on page 66.

Statement Example

The example finds the instruction which alters a specified storage location into the pattern (FEFE); it puts the system into a wait state when this event occurred.

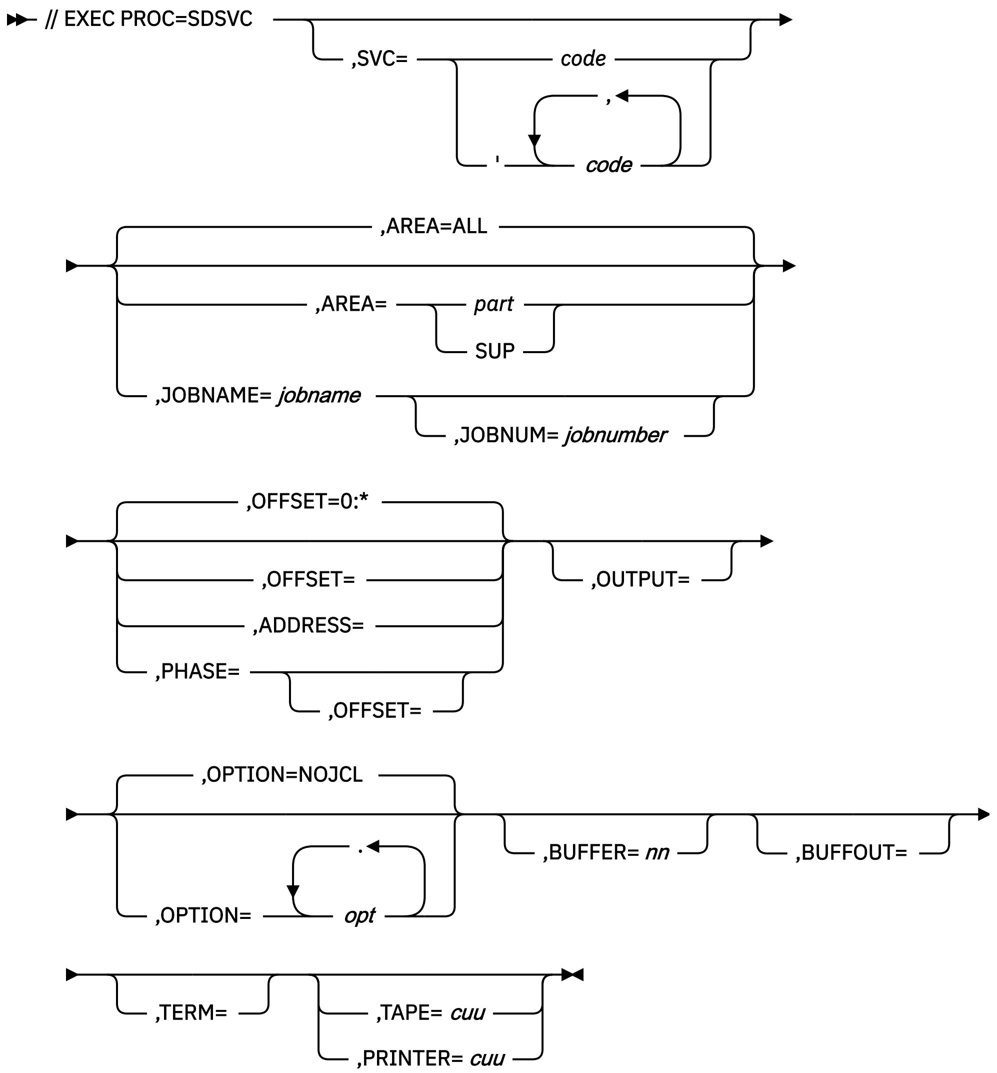
Here are the items of the trace setup shown below:

- Trace type: STORAGE
- Traced tasks: all tasks in the VSE system
- Address area whose contents alteration is traced: X'074754' up to X'074755'
- Alteration value to be traced: FEFE

- Additional trace definition: put the system into a wait state on the event of the defined storage alteration.
- Output destination: tape with device address 280
- Note that the continuation sign has to follow the comma if you use the console for input. If you use SYSIN to enter the procedure statement, the continuation sign has to be in column 72.

```
// EXEC PROC=SDSTOR,PATTERN=FEFE,AREA=ALL,ADDRESS='74754:74755',-
                    OPTION=HALT,TAPE=280
```

SVC Trace



See the “Additional Keyword Operands in Trace Procedure Statements” on page 129. The procedure SDSVC initializes traces which provides event records for all or specified SVC instructions. Define the SVC code in hexadecimal form.

For a description of the trace type and an example of the output, see “SVC Trace (Supervisor Call)” on page 67.

Defaults Set in the Procedure

If you omit SVC=, all SVC instructions are traced.

Statement Example

Here are the items of the trace setup shown below:

- Trace type: SVC
- SVC instruction defined by the SVC code: 3F
- Traced tasks: all tasks of the BG partition
- Traced storage area: BG partition area (SDAID default)
- Output destination: 10K bytes buffer
- Output device for buffer: tape with device address 280
- Event to write the buffer to tape: cancel or EOJ condition in the BG partition

```
// EXEC PROC=SDSVC,AREA=BG,SVC=3F,TAPE=280,BUFFER=10,BUFFOUT=CANCEL
```

Additional Keyword Operands in Trace Procedure Statements

When you initialize a trace using a procedure, the trace type which SDAID actually calls corresponds to the procedure name. The additional trace operands, for example the specification of the output device, correspond to the operands specified or defaulted in the procedure operands.

The additional operands which are specific for the trace initialization via procedures are described in this section. The other additional operands have been described in Chapter 8, “SDAID General Description,” on page 53.

The table shows all additional keyword operands in the format accepted in the EXEC PROC statement, a short description for each and a reference to their detailed description.

Table 12. Additional Keywords, Summary

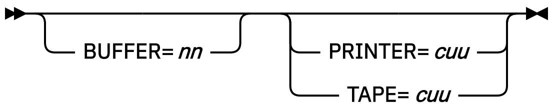
Operand	Function	See:
ADDRESS	Limit tracing to a certain address range	“Defining the Storage to be Traced: OFFset, ADDress, PHase, LTA” on page 70
AREA	Limit tracing to a certain system area	“Defining the Area to be Traced: AREA Definition” on page 69
JOBNAME[JOBNUM]	Limit tracing to a certain VSE/POWER job	“Defining the Job to be Traced: JOBNAME Definition” on page 70
OFFSET	Limit tracing to a partition or phase area	“Defining the Storage to be Traced: OFFset, ADDress, PHase, LTA” on page 70
PHASE	Limit tracing to a certain phase	“Defining the Storage to be Traced: OFFset, ADDress, PHase, LTA” on page 70
OPTION	Define additional trace options	“Defining the Trace Options: OPTion Definition” on page 80
OUTPUT	Define additional trace output	“Defining Additional Trace Output: OUTPut Definition” on page 72
UNIT	Define the device address	“Defining the Traced I/O Devices” on page 82
BUFFER|BU	Define the size of the output buffer	“BUFFER=, PRINTER=, TAPE=Keyword Operands” on page 130
BUFFOUT	Define the event to write the buffer	“BUFFOUT=Keyword Operand” on page 130

Table 12. Additional Keywords, Summary (continued)

Operand	Function	See:
TERM	Define the event which terminates the trace	"TERM=Keyword Operand" on page 131
PRINTER|P	Define the printer device address	"BUFFER=, PRINTER=, TAPE=Keyword Operands" on page 130
TAPE|T	Define the tape device address	"BUFFER=, PRINTER=, TAPE=Keyword Operands" on page 130

Define the Output Device in a Procedure Statement

BUFFER=, PRINTER=, TAPE=Keyword Operands



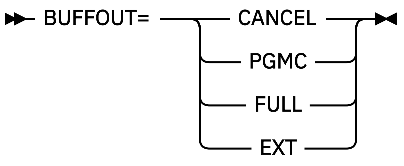
You define the output destination of the event trace records via the keyword operands BUFFER=nn, TAPE=cuu, or PRINTER=cuu.

Note: The abbreviations BU=nn, T=cuu, or P=cuu may be used.

BUFFER=nn
Defines the size of a wraparound buffer to collect the trace event records.

Note: The definition of a large wraparound buffer may cause a lack of SDAID storage. For information, see "Space Requirements for SDAID Execution" on page 50.

BUFFOUT=Keyword Operand



Via the BUFFOUT= keyword operand you define the condition which forces the write buffer operation.

BUFFOUT=CANCEL
Defines that the contents of the wraparound buffer is to be written to the output device (Printer or Tape) when a cancel or EOJ condition occurs.

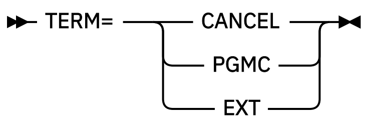
BUFFOUT=PGMC
Defines that the contents of the wraparound buffer is to be written to the output device (Printer or Tape) on any program check interruption (except page faults).

Note: If you specify BUFFOUT=CANCEL or BUFFOUT=PGMC, you must also specify the keyword operand AREA=partition-id or JOBNAME=.

BUFFOUT=FULL
Defines that the buffer is to be written to the output device whenever it is full.

BUFFOUT=EXT
Defines that the buffer is to be written to the output device whenever the external interrupt key is pressed.

TERM=Keyword Operand



TERM=CANCEL
: defines that tracing is to be terminated as soon as a cancel condition occurs in the traced partition.

TERM=PGMC
: defines that tracing is to be terminated as soon as a program check occurs in the traced partition.

TERM=EXT
: defines that tracing is to be terminated as soon as the external interrupt key is pressed.

Note: If TERM=CANCEL or TERM=PGMC is specified, AREA=partition-id or JOBNAME has to be specified, too.

Chapter 11. Initialize a Trace in Prompt Input Mode

This topic describes how you initialize an SDAID trace in prompt input mode. The prompt input mode works only in the attention routine.

Overview

You can set up SDAID traces in prompt mode, which operates in the attention routine.

You invoke the SDAID program in prompt mode by entering the attention routine (AR) command **SDAID** without another specification. Prompt mode is also activated if you process direct input mode commands in the attention routine with at least one prompt mode statement, like the question mark (?).

The trace output device is defined via prompts after you enter the **OUTDEV** command.

SDAID prompts you for the trace type when you enter the **TRACE** command in the AR.

You end the initialization process with the **READY** command.

Once you have initialized the SDAID trace, attention routine commands are used to start the trace execution (STARTSD), suspend it (STOPSD), and end it (ENDSD).

The trace output, an **event record,** is supplied for each occurrence of a traced event, according to your instructions.

You may request the event records to be written to a line printer, onto a magnetic tape, or into a wraparound buffer. The definition of the output device is given via the prompts following the OUTDEV command.

The prompts and the possible replies are shown in "Command Input Path Example" on page 136.

How to Initialize an SDAID Trace in Prompt Mode

SDAID trace initialization in prompt mode requires the commands shown in Table 13 on page 133:

Table 13. Input Command Summary

Command	Description	See:
SDAID	Attention routine command to invoke the SDAID program.	-
OUTDEV	Defines output device for the trace (printer, tape, or buffer).	"Output Device Definition in Prompt Mode: OUTDEV Command" on page 144
TRACE	Defines the event(s) to be traced. At least one TRACE command is required; up to ten may be submitted.	"Specifying the Trace: TRACE Command" on page 145
READY	Ends input of initialization commands OUTDEV and TRACE.	-

The Various SDAID Commands

SDAID prompts you for the output device of the trace when you enter **OUTDEV**.

One OUTDEV definition can be active in the system at one time. Any newly entered OUTDEV command overwrites the existing one.

Enter **TRACE** to be prompted by SDAID for the type(s) of traces you want. Up to ten 'TRACE' commands may be entered in one session.

You end the trace initialization in the attention routine with the **READY** command. When the READY command has been processed, no further OUTDEV or TRACE command can be entered.

Sample SDAID Trace Initialization

Figure 51 on page 134 shows a typical trace initialization session.

The session starts with the AR command ‘SDAID’. With the command ‘OUTDEV’ the output device is defined and the command ‘TRACE’ is entered to specify the trace type. The initialization process ends with the READY command.

Prompt Setup via the Attention Routine

```
→ sdaid
  4C05I PROCESSING OF ‘SDAID’ COMMAND SUCCESSFUL.
→ outdev
  4C08D SPECIFY OUTPUT DEVICE.+
→ tape
  4C08D SPECIFY PHYSICAL ADDRESS OF PRINTER/TAPE.+
→ 281
  4C05I PROCESSING OF ‘OUTDEV’ COMMAND SUCCESSFUL.
→ trace
  4C08D SPECIFY TRACE TYPE.+
→ inst
  4C08D SPECIFY OP-CODE(S) OR ‘*’ OR ‘BRANCH’.
→ *
  4C08D SPECIFY ONE OF THE KEYWORDS AREA OF JOBNAME.+
→ area
  4C08D SPECIFY TRACE AREA.+
→ ?
  ENTER SYSLOG-ID (LIKE BG OR F1)
  FOR TRACING A PARTITION, OR
  ‘SUP’ FOR THE SUPERVISOR, OT
  ‘ALL’ FOR TRACING ENTIRE SYSTEM
→ SUP
  4C08D SPECIFY TYPE OF LIMITS.+
→ add
  4C08D SPECIFY ADDRESS RANGE.+
→ 4000:7A00
  4C08D SPECIFY OUTPUT.+
→
  4C08D SPECIFY OPTIONS.+
→ nojcl
  4C08D SPECIFY OPTIONS.+
→
  4C05I PROCESSING OF ‘TRACE’ COMMAND SUCCESSFUL.
  1I40I READY.
→ ready
  4C05I PROCESSING OF ‘READY’ COMMAND SUCCESSFUL.
  I40I READY.
→ startsd
  4C05I PROCESSING OF ‘STARTSD’ COMMAND SUCCESSFUL.
  1I40I READY.
```

Figure 51. Example: Prompt Mode Trace Initialization

Notational Conventions

- SDAID messages (or help information) are shown in uppercase with a message number.
- Responses or commands for you to enter are shown in mixed case. In most responses a short form of the command is also allowed, and this is shown in uppercase. The non-mandatory part of the response is in lowercase. For example, the BRanch trace type specification can be abbreviated in the following way:

```
BR BRa BRan BRanc BRanc
```

How to Use Help and Cancel in Prompt Mode

- Messages for which you can request additional help information are indicated by a plus sign (+) at the end of the message.
- Request additional help by entering a question mark (?).
- You can cancel data entered for the current command by entering two question marks (??).

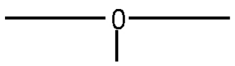
Figure 52 on page 135 shows how you can request help information and how the initialization process can be canceled.

```
→ trace
  4C08D SPECIFY TRACE TYPE.+
→ ?
  ENTER ONE OF THE FOLLOWING KEYWORDS:
   SVC       PGMCHECK   MONITOR    CANCEL
   INSTR                STORAGE    BRANCH
   PGMLOAD   EXTERNAL   BUFFER
   IO        SSCH       VTAMIO     VTAMBU
→ ??
  4D03I COMMAND CANCELED DUE TO USER REQUEST
```

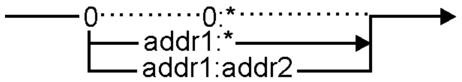
Figure 52. Example: Help and Cancel Initialization

How to Read the Following Prompting Mode Syntax Diagrams

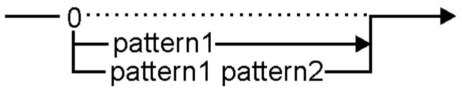
The following diagrams use a solid line, or a number of solid lines in parallel, as a specification path. Follow the line of the option that you select for your SDAID execution.



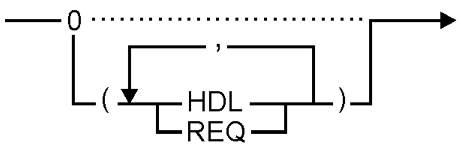
Indicates where you get a prompting message.



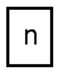
Three options, the dotted line indicating a default (in this case from 0 to end). Press ENTER for the default.



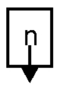
Same as above, but END/ENTER indicates no pattern instead of a default.



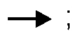
The comma (,) indicates that the optional operands can be entered more than once and in any sequence. Press ENTER when you have finished the input or none of the options is desired.



On-page connector.



Off-page connector (to a following page).



Command input completed.

Command Input Path Example

This section shows the prompt messages and the possible replies in the sequence of their processing.

Figure 53 on page 136 shows an example of the trace statement path. You can find the possible input in accordance to the prompt message 'SPECIFY TRACE TYPE' (BR, CA, ..). The example also indicates the prompt message after the reply 'inst' (SPECIFY OP * OR BR).

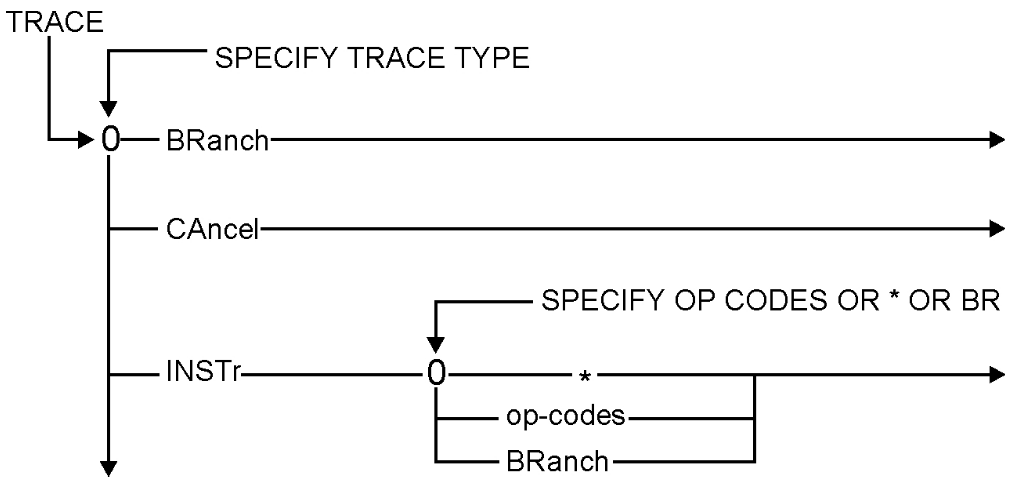


Figure 53. Sample Command Input Path

Command Input Paths

OUTDEV Command Input Path

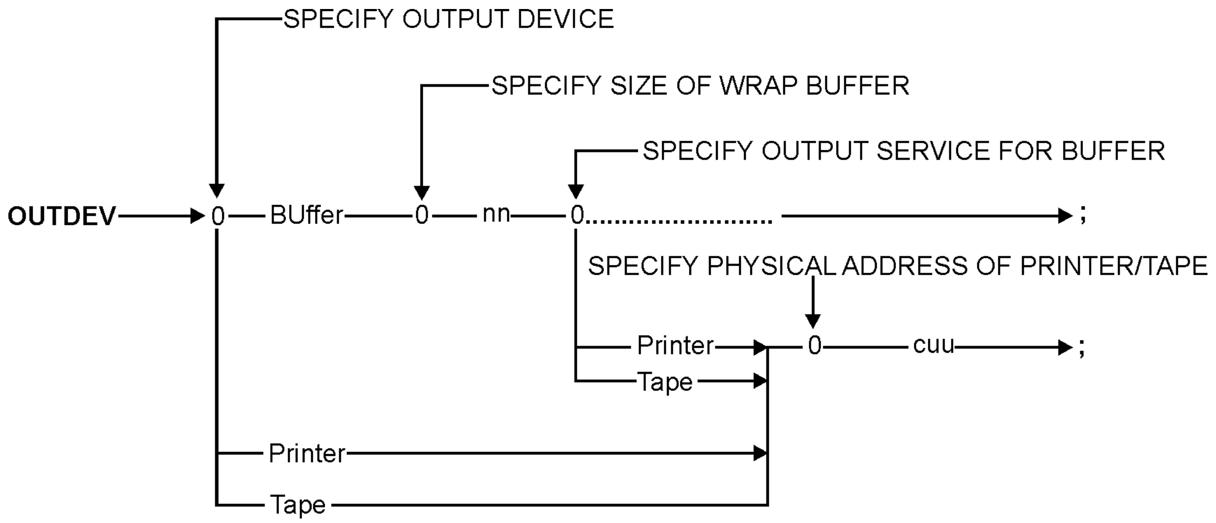

SPECIFY OUTPUT DEVICE
SPECIFY SIZE OF WRAP BUFFER
SPECIFY OUTPUT SERVICE FOR BUFFER
OUTDEV
0
BUffer
0
nn
0
;
SPECIFY PHYSICAL ADDRESS OF PRINTER/TAPE
Printer
Tape
0
cuu
;
Printer
Tape

Figure 54. OUTDEV Command: Syntax Diagram

TRACE Command Input Path

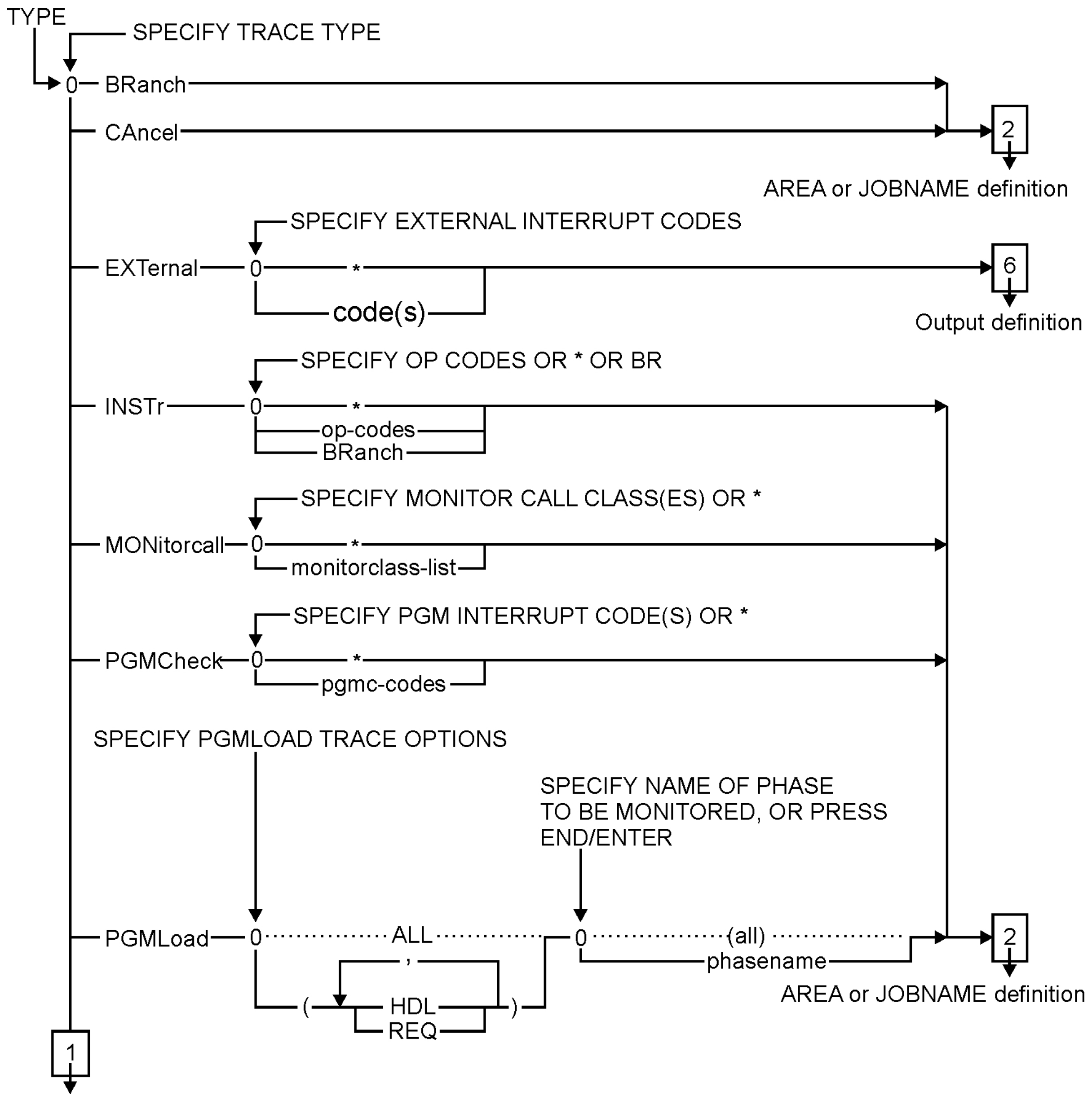

TYPE
SPECIFY TRACE TYPE
0
BRanch
CAncel
2
AREA or JOBNAME definition
SPECIFY EXTERNAL INTERRUPT CODES
EXTernal
0
*
code(s)
6
Output definition
SPECIFY OP CODES OR * OR BR
INSTr
0
*
op-codes
BRanch
SPECIFY MONITOR CALL CLASS(ES) OR *
MONitorcall
0
*
monitorclass-list
SPECIFY PGM INTERRUPT CODE(S) OR *
PGMCheck
0
*
pgmc-codes
SPECIFY PGMLOAD TRACE OPTIONS
SPECIFY NAME OF PHASE TO BE MONITORED, OR PRESS END/ENTER
PGMLoad
0
ALL
(
,
HDL
REQ
)
0
(all)
phasename
2
AREA or JOBNAME definition
1

Figure 55. TRACE Command: Syntax Diagram (1 of 7)

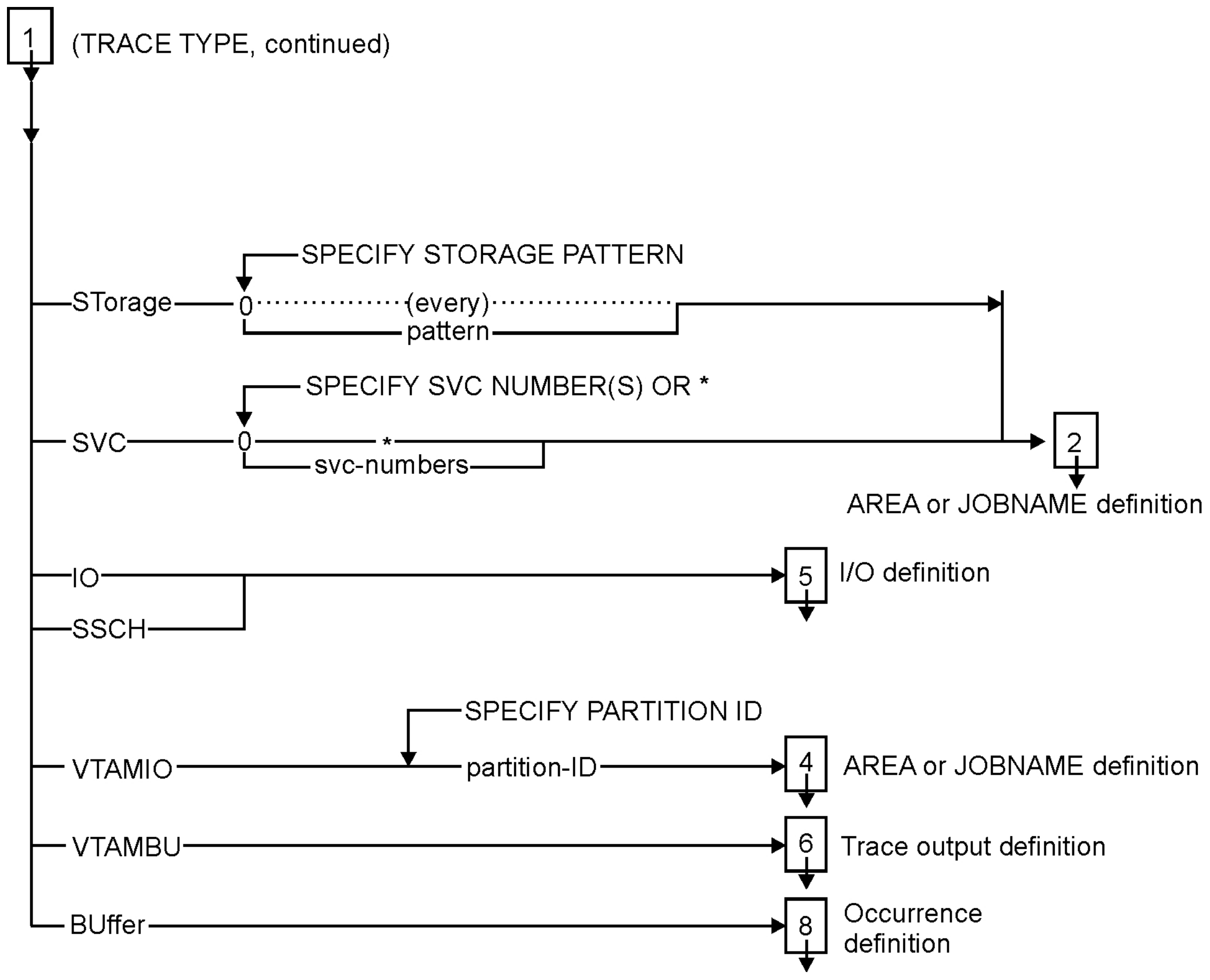


Figure 56. TRACE Command: Syntax Diagram (2 of 7)

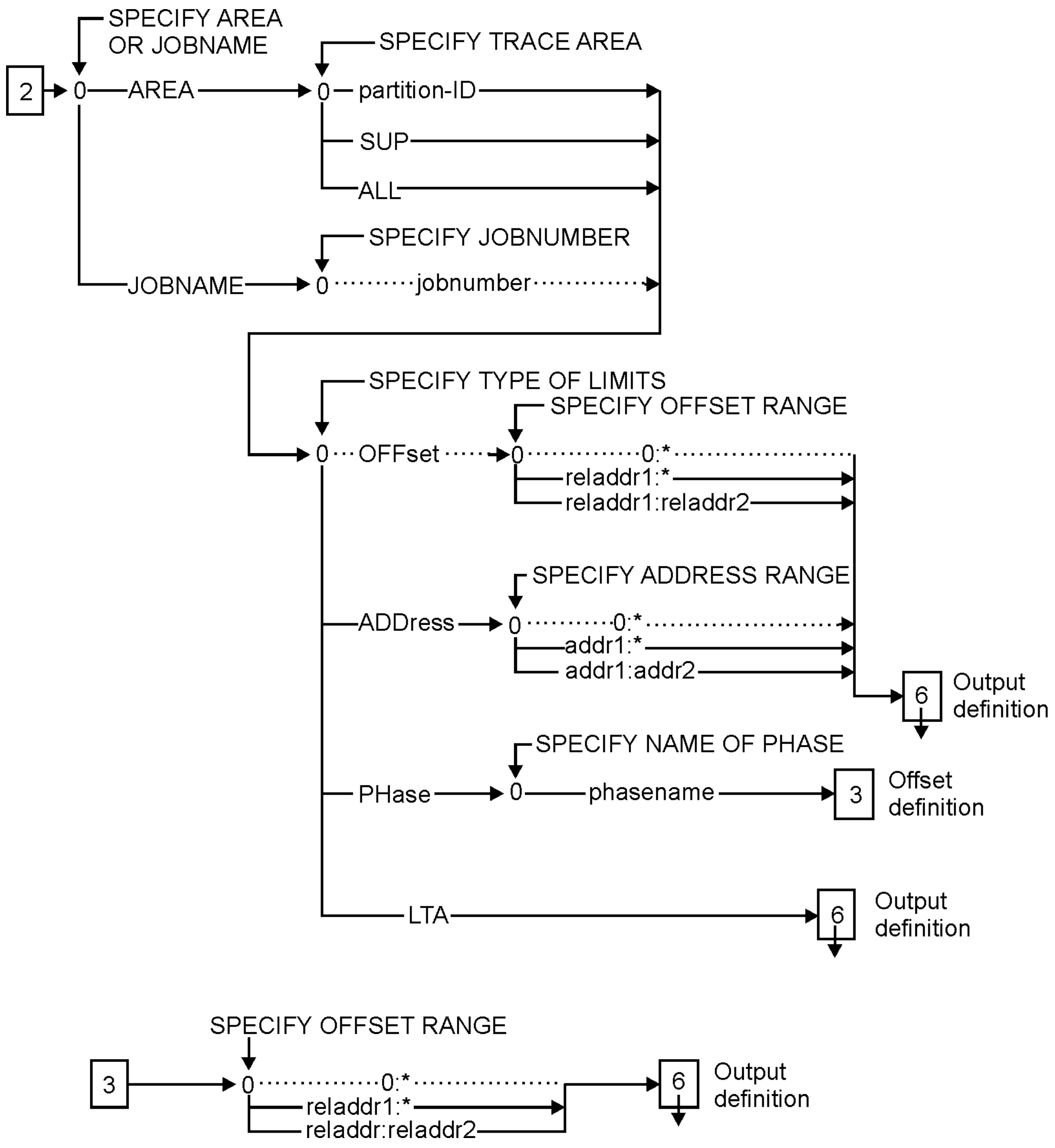


Figure 57. TRACE Command: Syntax Diagram (3 of 7)

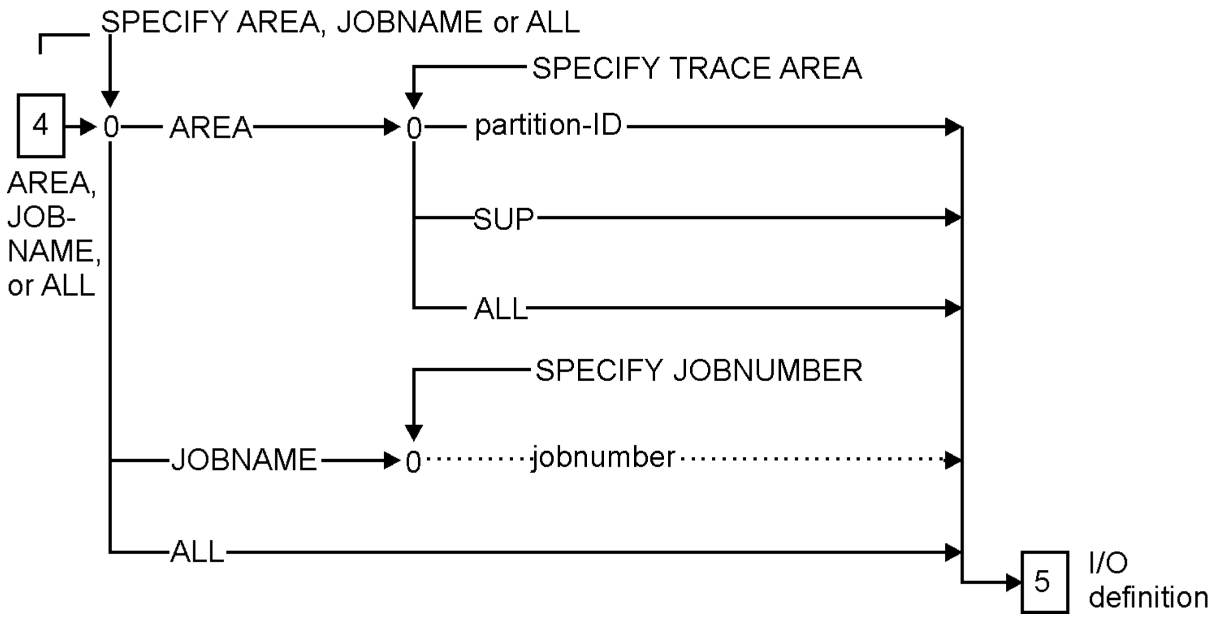


Figure 58. TRACE Command: Syntax Diagram (4 of 7)

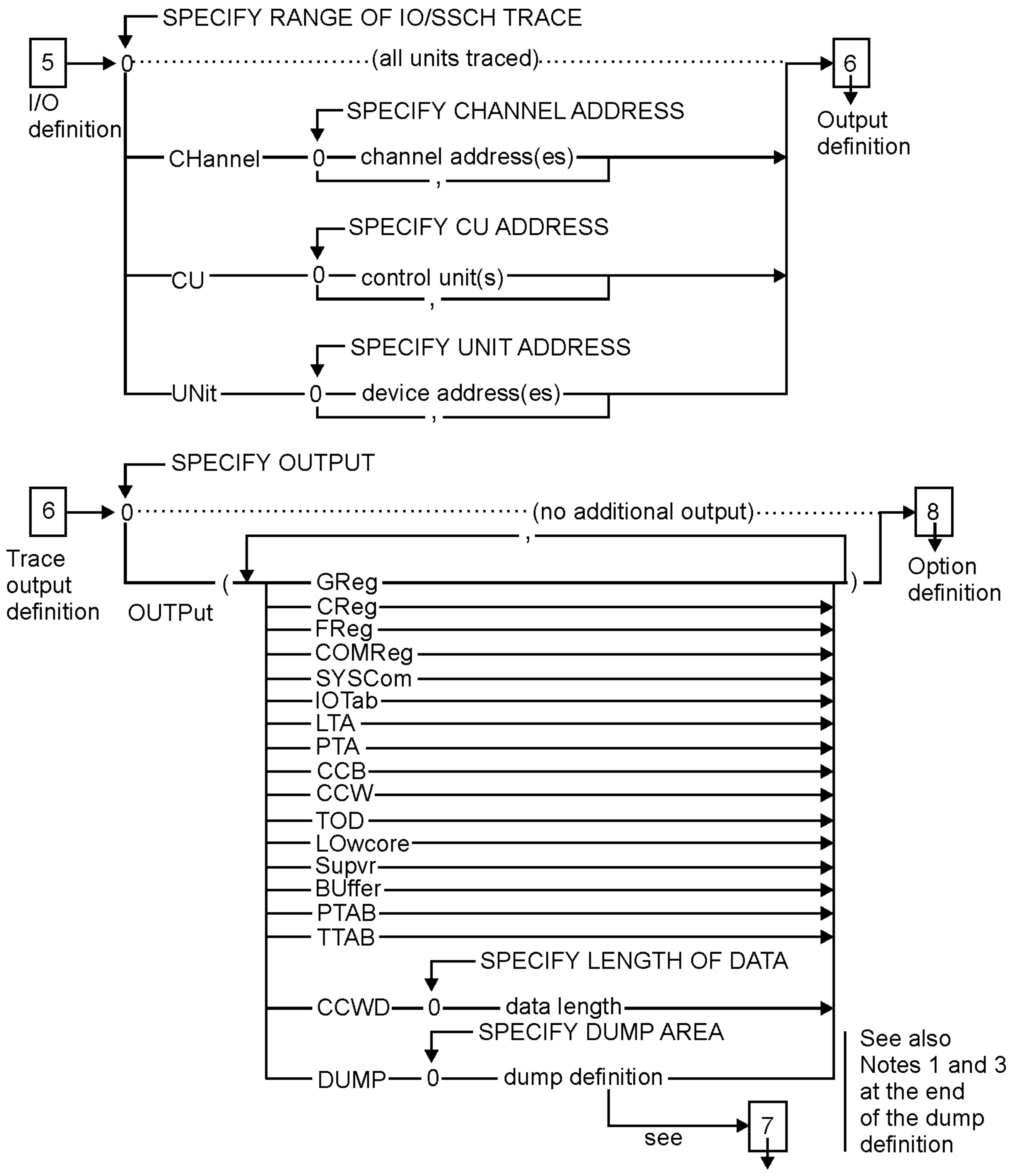
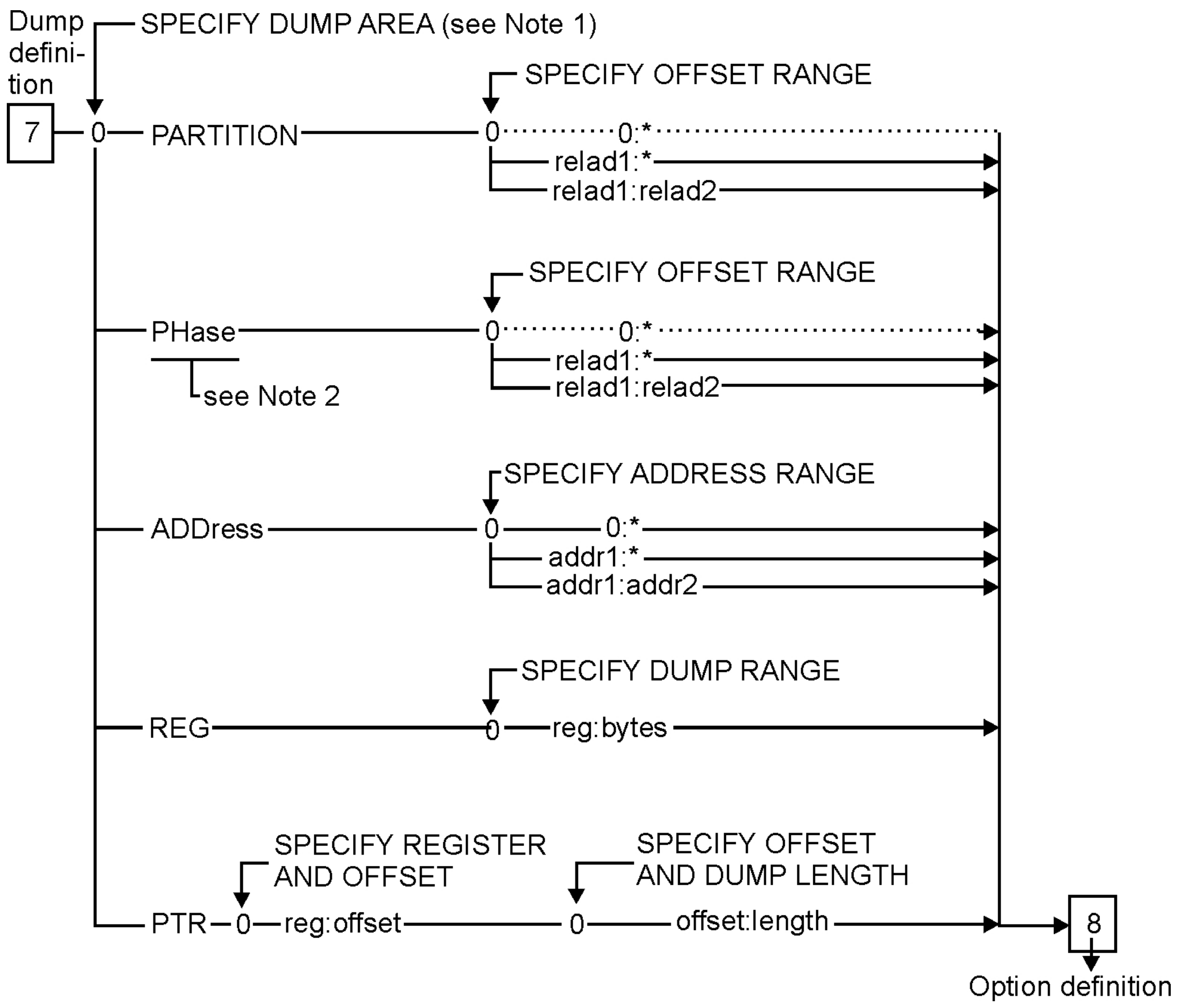


Figure 59. TRACE Command: Syntax Diagram (5 of 7)



Note:

1. Up to ten different areas may be specified with DUMP.
2. Can be specified only if a phase was previously defined in the area definition of the TRACE.
3. You need not specify the word OUTPut in prompt mode. SDAID prompts you for the definition of the additional output.

Figure 60. TRACE Command: Syntax Diagram (6 of 7)

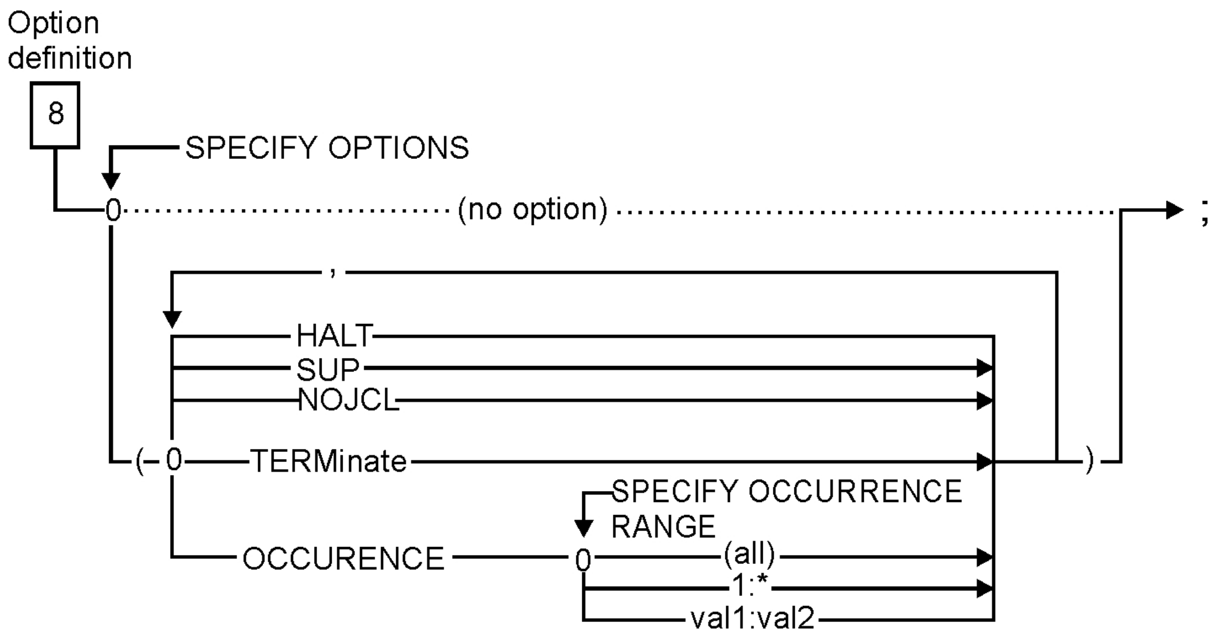


Figure 61. TRACE Command: Syntax Diagram (7 of 7)

Output Device Definition in Prompt Mode: OUTDEV Command

This section shows promptings and possible replies for the trace output device definition. Detailed information about the output device is given under "Defining the Output Device" on page 53.

When you enter the OUTDEV command, SDAID prompts you for control information as follows:

```
outdev
4C08D SPECIFY OUTPUT DEVICE.+
```

As a response to this prompting message, enter one of the following:

Printer
If printer is specified, the event records are written to a line printer at the time the particular event occurs.

Tape
When tape is defined, the trace records are written to tape in the form of 3K bytes blocks.

SDAID prompts you for the address of the output device in the following way:

```
4C08D SPECIFY PHYSICAL ADDRESS OF PRINTER|TAPE
```

BUffer
Writes the trace output to a wraparound buffer. SDAID prompts you for the size of the buffer as follows:

```
4C08D SPECIFY SIZE OF WRAP BUFFER.+
```

Enter the desired size of the buffer in number of blocks of 1K byte.

Possible Buffer Sizes

The possible buffer sizes depend on the output device for the buffer which is defined next.

Table 14. Buffer Sizes

Buffer to Printer or no output device	4K - 256K
Buffer to Tape	4K - 32K

Now, SDAID prompts you as follows:

```
4C08D SPECIFY OUTPUT DEVICE FOR BUFFER.+
```

Respond with either **Printer, Tape, or END/ENTER**.

Pressing END/ENTER causes no output device being defined.

Specifying the Trace: TRACE Command

Once you enter the command: **TRACE**, SDAID prompts you for the following control information:

Trace-type definition:
The type of event to be traced. See "Defining the Trace Type" on page 145.

Area definition:
The range of the trace in storage. See "AREA Definition" on page 157.

I/O definition:
Limits a trace operation to one or more channels, control units, or devices. See "I/O Definition" on page 159.

Output definition:
Additional (optional) trace information that is required to analyze the particular problem. See "Additional Output Definition" on page 160.

Option definition:
An option to:

- Stop system execution when the specified trace event occurs.
- Discontinue tracing when the specified trace event occurs.
- Avoid tracing of Job Control phases.
- Discontinue tracing when a defined number of events has been exceeded.
- Include supervisor routines into a partition trace.

See "Option Definition" on page 161.

You will find sample event records and a description of most of the trace types under "Summary of TRACE Types" on page 55.

Defining the Trace Type

This section, and the descriptions of the various trace types that follow, show the promptings and the possible replies when defining an SDAID trace.

Detailed information about the various trace types is given under "Summary of TRACE Types" on page 55.

You start the definition of your trace with the trace command in the attention routine.

```
trace
4C08D SPECIFY TRACE TYPE.+
```

Respond to the prompting message with any of the available trace types. For example if you want to initialize a branch trace, the response would look like this:

```
trace
4C08D SPECIFY TRACE TYPE.+
branch      ◄────────────  Your response
```

The SDAID then prompts you for additional information.

Please find a summary of the SDAID trace types in Table 15 on page 146.

Summary of Trace Types

Table 15 on page 146 gives the following information:

- The trace types shown in the format they can be entered. Note, that the uppercase letters indicate the shortest possible abbreviation.
- A short description of the trace type. All trace types are described in more detail under "Summary of TRACE Types" on page 55.
- A reference to the format description of the trace type.

Table 15. Trace Type Summary

Trace Type	Provides a Trace of:	See:
BRanch	Successfully executed branch instructions	"BRanch Trace" on page 147
BUffer	The trace buffer when it is full	"BUffer Trace" on page 147
CAncel	Program (main task) cancel or EOJ	"CAncel Trace" on page 147
EXTernal	External interrupts	"EXTernal (External Interrupt) Trace" on page 147
GETVIS	Getvis / Freevis requests	"GETVis (Getvis / Freevis Request) Trace" on page 148
INSTruction	Selected or all instruction(s) execution	"INSTruction (Instruction Execution) Trace" on page 149
IO	I/O interrupts	"IO (I/O Interrupt) Trace" on page 149
LOCK	Lock / Unlock requests of resources	"LOCK (Lock / Unlock of Resources) Trace" on page 150
MONitorcall	MC instructions	"MONitorcall Trace" on page 151
OSAX	OSAX adapter	"OSAX Adapter Trace" on page 151
PGMCheck	Program checks	"PGMCheck (Program Check) Trace" on page 152
PGMLoad	Phase load requests, or actual load	"PGMLoad (Program Load) Trace" on page 152
SSCH	Start Subchannel instructions	"Start Subchannel Instruction Trace" on page 154
STorage	Storage alterations	"STorage Alteration Trace" on page 154
SVC	Executed supervisor calls	"SVC (Supervisor Call) Trace" on page 154
VTAMBU	Usage of VTAM buffers	"VTAMBU (VTAM Buffer) Trace" on page 155
VTAMIO	VTAM I/O operations	"VTAMIO (VTAM I/O) Trace" on page 155
XPCC	XPCC communication actions	"XPCC (Partition Communication) Trace" on page 155

BRanch Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
branch       ◄────────────    Your response
```

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORDS AREA OR JOBNAME.+
```

See “AREA Definition” on page 157 and “JOBNAME Definition” on page 158.

BUffer Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
buffer       ◄────────────    Your response
```

The SDAID prompts you for the OCCurrence definition as next. For the format of these definitions. see “Option Definition” on page 161.

CAncel Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
cancel       ◄────────────    Your response
```

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORDS AREA OR JOBNAME.+
```

See “AREA Definition” on page 157 and “JOBNAME Definition” on page 158.

EXTernal (External Interrupt) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
external     ◄────────────    Your response
```

The SDAID prompts you for additional control information until you press END/ENTER, as follows:

```
4C08D SPECIFY TYPE OF EXTERNAL INTERRUPT OR *.+
```

Your response may be one to 8 of the following:

*****	To trace all types of external interrupts.
0040	To trace only key interrupts.
1003	To trace TOD-clock sync check.
1004	To trace clock comparator.
1005	To trace CPU timer.
1200	To trace malfunction alert.

1201	To trace emergency signal.
1202	To trace external call.
2401	To trace service signal.
2402	To trace logical device._* z/VM CP
2603	To trace PFAULT handshaking._* z/VM CP
4000	To trace IUCV, APPC._* z/VM CP
4001	To trace VMCF._* z/VM CP
END/ENTER	To continue.

The SDAID now prompts you for the definition of the OUTPUT.

GETVis (Getvis / Freevis Request) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
getv        <-----------  Your response
```

The SDAID prompts you for additional control information, as follows:

```
4C08D SPECIFY GETVIS/FREEVIS STORAGE REGION.+
```

Your response may be one of the following:

PARtition
Writes an event record if the requested or released space is within the partition Getvis area.

SPAce
Writes an event record if the requested or released space is within the dynamic partition Getvis area.

SVA
Writes an event record if the requested or released space is within the SVA.

SDAID then prompts you for a specific subpool name.

```
4C08D SPECIFY SUBPOOL NAME.+
```

Your response may be one of the following:

Press ENTER
Writes an event record for all subpools.

subpool_name
Writes an event record for the specific subpool only.

See "GETVIS / FREEVIS Trace" on page 58 for details of a subpool name format.

SDAID then prompts you for the LOCation of a Getvis / Freevis request:

```
4C08D SPECIFY LOCATION FOR GETVIS/FREEVIS REQUESTS.+
```

Your response may be one of the following:

Press ENTER
Writes an event record for all Getvis/Freevis requests.

BELow
Writes an event record for requests within the 24-bit Getvis area.

ANY
Writes an event record for requests within the 24-bit and 31-bit Getvis area.

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORDS AREA OR JOBNAME.+
```

See “AREA Definition” on page 157 and “JOBNAME Definition” on page 158.

INSTruction (Instruction Execution) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
instr       <-----------    Your response
```

The SDAID prompts you for additional control information, as follows:

```
4C08D SPECIFY OP-CODE(S) OR *.+
```

Your response may be one of the following:

op code(s)
(one to eight) Entered as either one-byte or two-byte hexadecimal values. If you specify more than one operation code, separate them by one or more blanks or by a comma (with or without blanks).

asterisk (*)
Defines all op codes.

You can also specify the following:

BRanch
Defines that all branch instructions have to be traced regardless whether the branch has been taken or not.

Sample responses:

```
d2
18 41,58 40, 50 9608
*
BRanch
```

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORD AREA OR JOBNAME.+
```

See “AREA Definition” on page 157 and “JOBNAME Definition” on page 158.

IO (I/O Interrupt) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
io          <-----------    Your response
```

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORD AREA, JOBNAME OR ALL.+
```

See “AREA Definition” on page 157 and “JOBNAME Definition” on page 158. ALL means all tasks of the system.

SDAID then prompts you for the definition of the specific I/O channel(s), control unit(s), or unit(s), as described under “I/O Definition” on page 159.

LOCK (Lock / Unlock of Resources) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
getv          <-----------   Your response
```

The SDAID prompts you for a specific resource name:

```
4C08D SPECIFY RESOURCE NAME OR PRESS END/ENTER.+
```

Your response may be one of the following:

Press ENTER
Writes an event record for all resources.

resource_name
Writes an event record for the specific resource only.

See "LOCK / UNLOCK Trace" on page 60 for details of the resource-name formats.

SDAID then prompts you for the type (LOCK or UNLOCK):

```
4C08D SPECIFY TYPE OF REQUEST OR PRESS END/ENTER.+
```

Your response may be one of the following:

Press ENTER
Writes an event record for all types of request.

Lock
Writes an event record for resource locking only.

Unlock
Writes an event record for resource unlocking only.

SDAID then prompts you for the scope (INTERNAL or EXTERNAL):

```
4C08D SPECIFY THE SCOPE OR PRESS END/ENTER.+
```

Your response may be one of the following:

Press ENTER
Writes an event record for all scopes.

INTernal
Writes an event record for Internal Locks or Unlocks.

EXTernal
Writes an event record for External Locks or Unlocks.

SDAID then prompts you for a volume ID:

```
4C08D SPECIFY VOLUME-ID OR PRESS END/ENTER.+
```

Your response may be one of the following:

Press ENTER
Writes an event record for all volume IDs.

volume_id
Writes an event record for the specified volume ID only.

SDAID then prompts you for a return code(s):

```
4C08D SPECIFY RETURN CODE(S) OR '*'.+
```

Your response may be one of the following:

*****
: Writes an event record for all return codes.

return_code(s)
: Writes an event record for the specified return code(s) only.

See "LOCK / UNLOCK Trace" on page 60 for details of how to specify return code(s).

SDAID then prompts you for the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORDS AREA OR JOBNAME.+
```

See "AREA Definition" on page 157 and "JOBNAME Definition" on page 158.

MONitorcall Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
monitorcall ◄────────────  Your response
```

SDAID prompts you for additional control information as follows:

```
4C08D SPECIFY MONITOR CALL CLASS(ES) OR *.+
```

Your response may be one of the following:

monitor classes
: defines the MC instructions to be traced by one or up to eight monitor classes.

  Monitor classes must be specified as one-digit hexadecimal values. If you specify two or more classes, separate them by one or more blanks, or by a comma with or without blanks.

  You may specify any valid monitor class; however, SDAID ignores a specification of class 2.

asterisk (*)
: defines all classes except class 2.

Sample responses:

```
3 5, 8,c d e f
a
*
```

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORDS AREA OR JOBNAME.+
```

See "AREA Definition" on page 157 and "JOBNAME Definition" on page 158.

OSAX Adapter Trace

Note: the output from an OSAX adapter trace is complex and normally only suitable for use by IBM personnel. Typically, an OSAX adapter trace would be requested by IBM support personnel after a related problem has been reported to them.

```
trace
4C08D SPECIFY TRACE TYPE.+
osax        ◄────────────  Your response
```

SDAID prompts you for additional control information as follows:

```
4C08D SPECIFY DATE PATH ADDRESS OR PRESS END/ENTER.+
```

Your response may be one of the following:

datapath
defines the data path address of an OSAX adapter, where *datapath* is a hexadecimal address of between 1 and 3 characters.

Press Enter
will process the data paths for all OSAX adapters.

SDAID then prompts you to reply if you require additional detailed output:

```
4C08D SPECIFY EXTENDED OUTPUT.+
```

Your response may be one of the following:

Yes
additional output will be provided.

No
no additional output will be provided.

Press Enter
Default: no additional output will be provided.

PGMCheck (Program Check) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
pgmcheck     ←————————    Your response
```

SDAID prompts you for additional control information as follows:

```
4C08D SPECIFY PROGRAM INTERRUPT CODE(S) OR '*'.+
```

Your response may be one of the following:

Program interrupt codes
(one to 16) must be specified in hexadecimal notation, leading zeros may be omitted. An asterisk (*) indicates all program check interruption codes - except those page or segment translation exceptions which are caused by the temporary absence of a storage page. The specification 10 11 traces all page or segment translation exceptions.

If you specify more than one program interrupt code, separate them by one or more blanks, or by a comma with one or more blanks.

```
Sample specifications:    1 13,05, 10, 0A
                          9
                          *
```

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORDS AREA OR JOBNAME.+
```

See "AREA Definition" on page 157 and "JOBNAME Definition" on page 158.

PGMLoad (Program Load) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
pgmload      ←————————    Your response
```

SDAID prompts you for additional control information as follows:

```
4C08D SPECIFY PGMLOAD TRACE OPTIONS.+
```

Your response may be one of the following:

Press ENTER
Writes an event record for all program load events (phase load, fetch request, or actual phase-load operation) within the specified trace range.

req
Writes an event record each time loading/fetching a phase is requested (see Notes 1 and 3 below).

hdl
Writes an event record each time a phase load/fetch request is handled; that is, when a requested phase is actually loaded into storage for execution (see Notes 1 and 3 below).

all
Writes an event record each time a phase load/fetch request occurs, and also each time a phase is actually loaded into storage for execution. This is the default (see Notes 1 and 2 below).

Note:

1. When you have entered a response to the above prompting message, SDAID repeats the prompting message until you respond by pressing ENTER.
2. If you want all program-load events to be traced, respond by pressing ENTER when SDAID displays the above prompting message *for the first time*.
3. If you want only one phase to be traced, submit the name of this phase after specifying HDL, REQ or ALL.

SDAID prompts you for additional control information as follows:

```
4C08D SPECIFY NAME OF PHASE TO BE MONITORED, OR PRESS ENTER
```

Your response may be one of the following:

Press ENTER
Defines all phases to be traced.

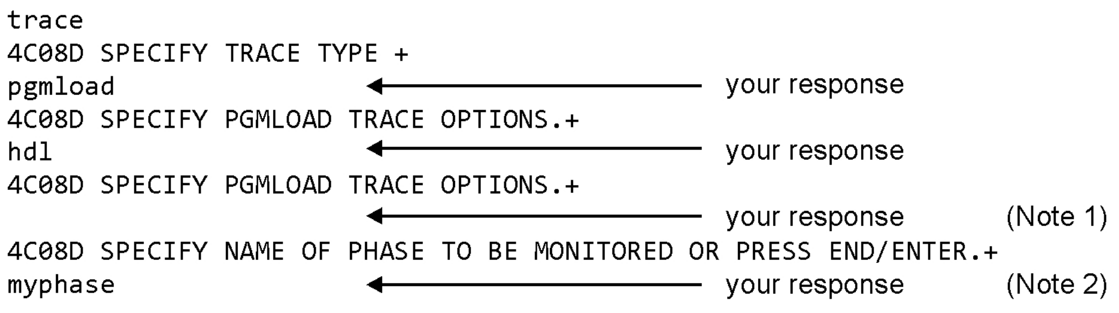
phase name
Defines the phase to be traced.

Figure 62 on page 153 is an example of a prompting sequence for a program load trace request.

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORDS AREA OR JOBNAME.+
```

See "AREA Definition" on page 157 and "JOBNAME Definition" on page 158.



```
trace
4C08D SPECIFY TRACE TYPE +
pgmload                 <--------------------  your response
4C08D SPECIFY PGMLOAD TRACE OPTIONS.+
hdl                     <--------------------  your response
4C08D SPECIFY PGMLOAD TRACE OPTIONS.+
                        <--------------------  your response      (Note 1)
4C08D SPECIFY NAME OF PHASE TO BE MONITORED OR PRESS END/ENTER.+
myphase                 <--------------------  your response      (Note 2)
```

Notes:
1. Indicates that no further control information of this type is to be entered.
2. Restricts the trace to the loading of the named phase.

Figure 62. Prompting for a PGMLoad Request

Start Subchannel Instruction Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
ssch        ◄────────────   Your response
```

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORDS AREA, JOBNAME OR ALL.+
```

See "AREA Definition" on page 157 and "JOBNAME Definition" on page 158. ALL means all tasks of the system.

SDAID further prompts you for the definition of the specific I/O channel(s), control unit(s), or unit(s), as described under "I/O Definition" on page 159.

STorage Alteration Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
storage     ◄────────────   Your response
```

SDAID then prompts you for additional control information as follows:

```
4C08D SPECIFY STORAGE PATTERN.+
```

Your response to this prompting message may be either of the following:

Press ENTER
: Requests an event record to be written whenever storage within the trace range *is altered.*

hexvalue
: Requests an event record to be written whenever storage within the trace range is *set to the specified value* (any hexadecimal value of up to four bytes). If you specify an odd number of digits, a zero is inserted to the left of the first specified hexadecimal digit.

Note: This option traces only program-altered storage, not that altered by I/O operations.

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORD AREA OR JOBNAME.+
```

See "AREA Definition" on page 157 and "JOBNAME Definition" on page 158.

Define ALL if you want to get all tasks of your system watched.

SVC (Supervisor Call) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
svc         ◄────────────   Your response
```

SDAID then prompts you for additional control information as follows:

```
4C08D SPECIFY SVC NUMBER(S) OR '*'.+
```

Your response to this prompting message may be either of the following:

one to 16 SVCs
Specify the SVC number in hexadecimal notation. If you specify two or more SVC numbers, they must be separated by one or more blanks, or by a comma with or without blanks.

asterisk (*):
Defines all SVC instructions to be traced.

Sample SVC specifications:

```
02 9,A 26
25
*
```

SDAID then prompts you for the definition of the trace area or the job name:

```
4C08D SPECIFY ONE OF THE KEYWORDS AREA OR JOBNAME.+
```

See “AREA Definition” on page 157 and “JOBNAME Definition” on page 158.

VTAMBU (VTAM Buffer) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
vtambu      <-----------   Your response
```

The SDAID prompts you for the OUTPut definition. See “Additional Output Definition” on page 160.

VTAMIO (VTAM I/O) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
vtamio      <-----------   Your response
```

SDAID then prompts you for the definition of the partition-ID as shown below:

```
4C08D SPECIFY PARTITION ID.+
```

Specify the partition where VTAM is running, F3 for example.

SDAID then prompts you next for the definition of the specific I/O channel(s), control unit(s), or unit(s), as described under: “I/O Definition” on page 159.

XPCC (Partition Communication) Trace

```
trace
4C08D SPECIFY TRACE TYPE.+
xpcc        <-----------   Your response
```

The SDAID prompts you for a specific application name:

```
4C08D SPECIFY APPLICATION NAME OR ‘*’.+
```

Your response may be one of the following:

*****
Writes an event record for all applications.

application_name
Writes an event record for the specific application only.

SDAID then prompts you for a specific to_application name:

```
4C08D SPECIFY TO_APPLICATION NAME OR '*'.+
```

Your response may be one of the following:

*****
: Writes an event record for all to_applications.

to_application_name
: Writes an event record for the specific to_application only.

SDAID then prompts you for an xpcc function:

```
4C08D SPECIFY XPCC FUNCTION.+
```

Your response may be one of the following:

*****
: Writes an event record for all xpcc applications.

xpcc_function
: Writes an event record for the specified xpcc_function only.

SDAID then prompts you for the direction of tracing:

```
4C08D SPECIFY DIRECTION OF TRACING.+
```

Your response may be one of the following:

Press ENTER
: Writes an event record for incoming and outgoing xpcc requests.

in
: Writes an event record for incoming xpcc requests.

out
: Writes an event record for outgoing xpcc requests.

both
: Writes an event record for incoming and outgoing xpcc requests.

SDAID then prompts you for a specific return code:

```
4C08D SPECIFY DEDICATED RETURN CODE.+
```

Your response may be one of the following:

*****
: Writes an event record for all return codes.

return_code
: Writes an event record for the specified return code only.

SDAID then prompts you for a comparator (SUSR / IJBXSUSR):

```
4C08D SPECIFY COMPARATOR TO COMPARE SUSR AGAINST IJBXSUSR.+
```

Your response may be one of the following:

nocomp
: No comparison is required.

EQ | NE | GT | GE | LT | LE

If your response is *not* **nocomp**, SDAID prompts you for a hex value which will be compared against the content of IJBXSUSR.

```
4C08D SPECIFY SUSR- HEXVALUE.+
```

See “XPCC Trace” on page 69 for details of how to prepare a compare field.

SDAID then prompts you for a comparator (RUSR / IJBXRUSR):

```
4C08D SPECIFY COMPARATOR TO COMPARE RUSR AGAINST IJBXRUSR.+
```

Your response may be one of the following:

nocomp
: No comparison is required.

EQ | NE | GT | GE | LT | LE

If your response is *not* **nocomp**, SDAID prompts you for a hex value which will be compared against the content of IJBXRUSR.

```
4C08D SPECIFY RUSR-HEXVALUE.+
```

See “XPCC Trace” on page 69 for details of how to prepare a compare field.

SDAID then prompts you for an XPCC from_area:

```
4C08D SPECIFY XPAREA(FROM-PARTITION).+
```

Your response may be one of the following:

Press ENTER
: Writes an event record for all from_areas.

syslog_id
: Writes an event record for the specified from_area only.

SDAID then prompts you for an XPCC to_area:

```
4C08D SPECIFY XPTOAREA(TO-PARTITION).+
```

Your response may be one of the following:

Press ENTER
: Writes an event record for all to_areas.

syslog_id
: Writes an event record for the specified to_area only.

AREA Definition

This section describes the SDAID promptings and gives some examples of the possible replies to the ARea Definition. More detailed information about the ARea definition and the corresponding storage region definitions is given in the following sections:

- “Defining the Area to be Traced: AREA Definition” on page 69.
- “Defining the Storage to be Traced: OFFset, ADDress, PHase, LTA” on page 70.

SDAID prompts you for the required area definition by displaying the message:

```
4C08D SPECIFY TRACE AREA.+
```

Enter one of the following responses:

- partition-ID
- SUP
- ALL

JOBNAME Definition

The JOBNAME (and JOBNUMBER) definition allows you to trace a VSE/POWER job in a dynamic or static partition. You can use either the AREA or the JOBNAME definition, but not both. For details about the JOBNAME definition, refer to "Defining the Job to be Traced: JOBNAME Definition" on page 70.

The possible prompts are the same as for the AREA definition.

SDAID prompts you for the optional JOBNUMBER definition by displaying the message:

```
4C08D SPECIFY JOBNUMBER.+
```

Enter the job number of the VSE/POWER job to be traced.

Prompts after AREA and JOBNAME Definitions

SDAID prompts you for the definition of the storage area to be traced as follows:

```
4C08D SPECIFY TYPE OF LIMITS.+
```

Your response to this prompting message is one of the following:

- END/ENTER
- OFFset
- PHase
- ADDress
- LTA

Press ENTER
To trace the requested events within the storage occupied by the defined partition.

OFFset
SDAID prompts you for the actual offset values:

```
4C08D SPECIFY OFFSET RANGE.+
```

Your response to this message, a pair of offsets, is discussed below, under "PHase".

Note that OFFset does not apply to ARea=All.

PHase
SDAID prompts for the phase name as follows:

```
4C08D SPECIFY NAME OF PHASE.+
```

Then SDAID prompts you for further limitation of the trace range:

```
4C08D SPECIFY OFFSET RANGE.+
```

Your response to this message (for either partition or phase offset) is one of the following:

ENTER
reladdr1:reladdr2
reladdr1:*

Press ENTER
to trace the events in the entire area allocated to the specified partition, supervisor or phase.

reladdr1:reladdr2
to define an address range in hexadecimal notation.

reladdr1:*
to define an address range starting with 'reladdr1' up to the end of the specified partition, supervisor or phase.

ADDress
SDAID prompts you for the actual address range:

```
4C08D SPECIFY ADDRESS RANGE.+
```

Your response to this message is either of the following:

- END/ENTER
- addr1:addr2
- addr1:*

Press ENTER
to trace the events of the tasks with the defined partition-id without address limitation.

addr1:addr2
to define a certain address area within the partition or supervisor.

addr1:*
to define an address area starting with 'addr1' up to the end of the partition or supervisor.

LTA
SDAID traces the events of the specified partition or supervisor which occur in the Logical Transient Area.

I/O Definition

The I/O definition limits the range of an **IO, SSCH, or VTAMIO** trace to one or more devices, to one or more control units, or to one or more channels. This section describes the SDAID promptings. Detailed information about the I/O definitions is provided under "Defining the Traced I/O Devices" on page 82.

SDAID prompts you for the definition as follows:

```
4C08D SPECIFY KEYWORD UNIT OR CU OR CHANNEL.+
```

Your response to this prompting message may be one of the following: (detailed descriptions follow)

- ENTER
- UNit
- CU
- CHannel

Press ENTER
To define all I/O devices.

UNit
SDAID prompts you for the hexadecimal specification of up to 8 unit addresses as follows:

```
4C08D SPECIFY UNIT ADDRESS(ES).+
```

If you specify more than one address, separate them by one or more blanks, or by a comma with one or more blanks or without a blank.

If you specify a 1-digit device address, SDAID assumes channel 0 and control unit 0; for a 2-digit device address, SDAID assumes channel 0.

Sample device-address list specifications:

```
003, e 181 281
282
```

```
e     (same as 00e)
0e    (same as 00e)
```

CU

SDAID prompts you for the hexadecimal definition of up to 16 control unit addresses as follows:

```
4C08D SPECIFY CONTROL UNIT ADDRESS(ES).+
```

If you specify more than one address, separate them by one or more blanks, or by a comma with one or more blanks or without a blank.

Sample control-unit address list specifications:

```
1, 2a 3f
1c
2    (same as 02)
```

CHannel

The program prompts you for one or up to 16 channel addresses as follows:

```
4C08D SPECIFY CHANNEL ADDRESS(ES).+
```

If you specify more than one address, separate them by one or more blanks, or by a comma with one or more blanks or without a blank.

Sample channel address specifications:

```
1
0 2,  3
```

Additional Output Definition

This section gives information on the various responses to the SDAID promptings. If you want more detailed information on the output definitions, refer to "Defining Additional Trace Output: OUTPut Definition" on page 72 and Table 5 on page 72.

SDAID prompts you to specify additional output in the following way:

```
4C08D SPECIFY OUTPUT.+
```

You may respond with the following output definitions:

BUffer	FReg	PTAB
CCB	GReg	SUPvr
CCW	IOTab	SYSCom
CCWD	LOCKTE	TOD
COMReg	LOwcore	TTAB
CReg	LTA	XPCCB
DUMP	PTA	XPDATABU

For each prompt, you may specify one definition. Prompting continues until you press ENTER without a definition. This ends the output definition.

Option Definition

This section gives information on the various responses to the SDAID promptings. If you want more detailed information on the option definitions, refer to "Defining the Trace Options: OPTion Definition" on page 80.

SDAID prompts you for an option definition as follows:

```
4C08D SPECIFY OPTIONS.+
```

You respond with one of the following:

Halt
NOJCL Specification
NOSource
NOTarget
OCcurrence Definition
SUPervisor
Termination Specification

If you define OCcurrence, SDAID prompts you as follows:

```
4C08D SPECIFY OCCURRENCE RANGE.+
```

Respond with:

ENTER
value1:value2

Press ENTER
To indicate that you want all occurrences of the specified event to be traced (same as if you defined 1:*).

value1:value2
To limit tracing (value2 must be higher than or equal to value1). See the examples below.

Sample occurrence definitions:

```
1:1     trace only the first occurrence
        of the event
1:*     trace all occurrences of the
        event (this is the default value)
5:12    trace selected occurrences
        (5 to 12) of the specified event
```

Chapter 12. Start/Stop and End the Trace

This topic describes how you can start, stop, or terminate an initialized SDAID trace, and how to control the trace under exceptional conditions.

The Required Commands

STARTSD/STOPSD Commands: Starting and Stopping

Once you have entered the READY command which ends the initialization process, you can activate the trace at once or later. To start or restart the trace operation, enter the command

```
►► STARTSD ►◄
```

without any operand.

Note: If the trace was stopped by an event itself (TERMinate specified with the TRACE command), the trace operation can be restarted by issuing the STOPSD command followed by the STARTSD command.

The STARTSD command is rejected if the interactive trace program is active for any partition.

To interrupt the trace operation with the restart capability retained, enter the command

```
►► STOPSD ►◄
```

without any operand.

Note: When a tape is defined as output device, every STOPSD or ENDSD command writes a tapemark on the tape if there was any trace event. If, for example, you specify three times STARTSD/STOPSD within an SDAID session, you get three trace files on your trace output tape. However, if there was no trace event since the last STARTSD command, the tape remains unchanged.

ENDSD Command: Ending Execution

You can end the SDAID session by issuing the command:

```
►► ENDSD ►◄
```

without any operands. The ENDSD command releases all resources that were used by the program during the session, including the storage space that was occupied by SDAID and closes the trace output device. You may enter this command at any time during a session.

Attention Routine Command Example

The example in Figure 63 on page 164 shows how an initialized SDAID trace is started, interrupted and ended. After the ENDSD command has been processed all of the initialized trace information is released.

```
 .
startsd ◊
4C05I PROCESSING OF 'STARTSD' COMMAND SUCCESSFUL
 .
 .
 .
stopsd ◊
4C05I PROCESSING OF 'STOPSD'  COMMAND SUCCESSFUL
 .
 .
 .
startsd ◊
4C05I PROCESSING OF 'STARTSD' COMMAND SUCCESSFUL
 .
 .
 .
endsd ◊
4C05I PROCESSING OF 'ENDSD'  COMMAND SUCCESSFUL
```

Figure 63. Attention Routine Commands to Start, Stop and End the Trace

How to Control the Trace under Exceptional Conditions

The start and stop procedures described above can be used only when the attention routine is available. When you try to start or stop a trace, the attention routine may be unavailable because of the problem you are trying to identify, or because SDAID is in a wait state.

This section tells you how to control traces:

- When the system is in an unintended loop;
- When a trace is running and the attention routine is not available;
- When the system is in a wait state.

Tracing an Unintended Loop

Perform the following steps to use the SDAID branch, instruction or storage-alteration traces to gather information about an unintended loop:

1. Initialize one of the trace types mentioned above in the normal way.
2. Start the trace with the STARTSD command.
3. Display the contents of control register 9 with the control processors alter/display feature.
4. Notice the contents of this control register for later use.
5. Set bits 0 through 3 to zeros with the alter display feature. This stops the trace.
6. Recreate the loop condition by submitting the same job mix that existed when the particular loop occurred the first time.
7. When the loop appears again, restart SDAID operation by setting those bits of control register 9 to a value of 1 which you have set to zeros before.

   ```
   Bit    Effect if set to 1
   ---    ------------------
    0     Successful branches are traced.
    1     Instruction executions are traced.
    2     Storage alterations are traced.
   ```

   (See "Hardware Alter/Display" on page 228 for information on how to use the Alter/Display feature.)

Control register A contains the start and control register B the end address of the trace. You may change this address range by varying the addresses stored in control registers A and B.

To resume operation after SDAID has collected sufficient information about the loop, and if you cannot exit from the loop, re-IPL VSE.

Terminating SDAID Program Without the Attention Routine

It may happen that you can no longer request the attention routine to gain control of your processor. At that point, SDAID operation cannot be stopped as usual by entering the command STOPSD. Instead you can perform the following steps:

If the trace type is INST, BR, or STORAGE, you can use the following method:

1. Change your processor's mode of operation to manual.
2. Alter bits 0 through 2 of control register 9 to zero (using the alter/display feature).
3. Let SDAID finish execution by changing your processor's mode of operation back to normal.

For more information on the values to be set into the control registers used by SDAID, consult the *Principles of Operation* manual pertaining to your processor.

If OUTDEV is a printer, the following method to stop the trace is possible:

1. Stop the printer device.
2. Wait until the system goes into the wait state.
3. Press the external interrupt key to stop the trace output.
4. Stop the trace with the STOPSD command.

Starting/Terminating Tracing in a System Wait Condition

In some cases SDAID forces a system wait condition. How you can restart the system by starting or terminating the trace options is the subject of this section.

Wait Due to OPTION=HALT

You may define that the system enters the wait state at occurrence of a specific event. This is accomplished by the option ‘OPTION=HALT’ defined together with the desired event.

When the system has entered the wait, the address part of the wait PSW contains the value X'00EEEE'. The following actions may be taken to get out of the wait state:

1. If you want to continue tracing:

   Press the external interrupt key once. The system will enter the wait state again on the next occurrence of the traced event.
2. If you want to continue tracing but without OPTION=HALT:

   Enter X'FF' in storage location zero,
   Press the external interrupt key.

This removes the OPTION=HALT specification. The system continues tracing but does not enter the wait state on the next occurrence of the same event again.

System Wait Due to Intervention Required at the Output Device

SDAID loads a wait PSW with the value of X'EEEEEE' in the address part. The required operator action is described under “Exceptional Conditions on the Output Device” on page 54.

Part 4. Info/Analysis

Info/Analysis is a tool for:

- Dump file management
- Problem source identification
- Problem analysis

Chapter 13. Info/Analysis: Introduction

With Info/Analysis, you can simplify the task of using dump data to solve software problems. Info/Analysis assists you in this task through the following functions:

- Dump Management - to list the dumps being managed by Info/Analysis, to add or delete dumps from that list, and to delete dumps from the system.
- Dump Symptoms - to display problem failure information collected by the dumping component and by subsequent analysis routines.
- Dump Viewing - to display dump data in hexadecimal and character format, format, and display control blocks and other dump data that may be pertinent to the problem, to invoke dump analysis routines, and to display the results of those routines.
- Dump Offload - to copy a dump to tape for later retrieval.
- Dump Onload - to copy a dump to a dump sublibrary.

Operating Environment

Info/Analysis runs in a z/VSE partition with a size of at least 1M of storage.

Info/Analysis processes storage dumps that result from errors within the system, subsystems or user programs running on the system. The dumps are created by system dump and Standalone Dump Programs. Info/Analysis does not directly access the dump data. Rather, it uses system facilities to retrieve and update dump data and the symptom record. The symptom record is a collection of problem-related information stored in the dump and its extensions.

Info/Analysis uses a dump management file to maintain information about dumps. A dump must be identified in this file before it can be processed by Info/Analysis. This file is maintained using the Dump Management function.

Info/Analysis also uses an external routines file. This file contains a list of analysis routines that you may invoke to process dump data. The file also identifies user exit routines and dump access routines called by Info/Analysis.

You may enter control statements in two modes:

- Line mode - from the operator console
- Reader mode - from the system input device reader defined as the input area

From a z/VSE partition, all output of batch operations is routed to the SYSLST device assigned to the partition. In line mode, messages are sent to the console as well as to SYSLST. SYSLST must always be assigned to a unit record device. When running in reader mode, SYSIPT must be assigned to a unit record device.

The Dump Management File

The dump management file BLNDMF contains information about dumps managed by Info/Analysis.

Info/Analysis adds this information either during dump management invocation (Info/Analysis searches the dump sublibraries for new dumps automatically), or when you specify the name of a new dump. For a dump produced as a result of a DUMP attention routine command or for a stand-alone dump you want to onload, supply a name via the Info/Analysis statement

```
DUMP NAME  (specify the current dump)
```

Once information about a dump has been added to the dump management file, the Info/Analysis functions can be used to process the dump.

A dump entry remains in the dump management file until the dump is deleted using the Info/Analysis function.

Initializing the Dump Management File

Before you can use the functions of Info/Analysis the dump management file has to be initialized.

This initialization is accomplished by the UTILITY statement of Info/Analysis. The statement is used at system installation time and whenever you want to initialize or recreate the dump management file, for example after you have increased the size of the file, or after the file has been damaged.

For an explanation of how to change the size of the dump management file see "UTILITY - Initialize Dump Management File" on page 179. Also refer to skeleton SKDMPINI in ICCF library 59.

Figure 64 on page 170 shows a job example to initialize the dump management file.

Sample Initialization Job

```
// JOB   INIT
// ASSGN SYSLST,00E
// ASSGN SYS020,252    Dump library
// ASSGN SYS016,252    Dump management file
// ASSGN SYS017,252    External routines file
// DLBL SYSDUMP,'VSE.DUMP.LIBRARY',,VSAM,                                   X
              CAT=IJSYSCT,                                                  X
              DISP=(OLD,KEEP)
// DLBL BLNDMF,'INFO.ANALYSIS.DUMP.MGNT.FILE',0
// EXTENT SYS016,SYSWK1,1,0,9030,15
// DLBL BLNXTRN,'INFO.ANALYSIS.EXT.RTNS.FILE',1999/365,SD
// EXTENT SYS017,SYSWK1,1,0,9045,15

// EXEC INFOANA,SIZE=300K


SELECT DUMP MANAGEMENT | use the
UTILITY                | utility
RETURN                 | function
                       |
SELECT END             |

/*
/&
```

Note: The dump library is located in VSAM space if initial installation was performed with z/VSE 5.1 or later.

Figure 64. Sample Job: Dump Management File Initialization

The External Routines File

The external routines file contains the names of dump analysis exit routines. These routines are used to analyze dumps stored in one of the dump sublibraries. The external routines file contains the name and, optionally a description of each routine available for use with Info/Analysis.

Presently the external routines file contains four name lines. DFHPD410 analyses a dump of the CICS Transaction Server partition. IJBXDBUG is the common analysis routine for stand alone dumps. IJBXSDA formats the SDAID buffer in a stand-alone dump. IJBXCSMG formats the console buffer in a stand-alone dump.

The name of the external routines file is BLNXTRN. The job INITDUMP.Z creates the external routines during the system build process. If the external routines file is damaged you may recreate it via a DITTO job (see Figure 65 on page 171), or if the DITTO program is not available in your system, via an OBJMAINT job (see Figure 66 on page 171).

Loading the Info/Analysis External Routines File

The sample jobs shown in Figure 65 on page 171 and Figure 66 on page 171 record the names of the analysis routines DFHPD410, IJBXCSMG, IJBXDBUG and IJBXSDA in the Info/Analysis external routines file

Sample Jobs of External Routines File

```
// JOB LOAD1
// DLBL BLNXTRN,'INFO.ANALYSIS.EXT.RTNS.FILE',2011/365,SD
// EXTENT SYS017,SYSWK1,1,0,9045,15
// UPSI 1
// EXEC DITTO
$$DITTO CSQ BLKFACTOR=1,FILEOUT=BLNXTRN
ANEXIT DFHPD410  CICS DUMP ANALYZER
ANEXIT IJBXCSMG  ANALYSE CONSOLE BUFFER
ANEXIT IJBXDBUG  ANALYSE STANDALONE DUMP ROUTINE
ANEXIT IJBXSDA   SDAID BUFFER FORMATTING ROUTINE
/*
$$DITTO EOJ
/*
/&
```

Note: The example shows the DITTO statements for an external routines file on CKD disk. If the external routines file is on an FBA disk the DITTO command line reads like $$DITTO CSQ BLKFACTOR=1,FILEOUT=BLNXTRN,CISIZE=512

Figure 65. Sample Job: Loading the External Routines File via DITTO

```
// JOB LOAD2
// ASSGN SYS004,00C
// ASSGN SYS005,SYSWK1
// DLBL UOUT,'INFO.ANALYSIS.EXT.RTNS.FILE',2011/365,SD
// EXTENT SYS005,SYSWK1,1,0,9045,15
// EXEC OBJMAINT
./ CARD DLM=$$
./ Copy
ANEXIT DFHPD410  CICS DUMP ANALYZER
ANEXIT IJBXCSMG  ANALYSE CONSOLE BUFFER
ANEXIT IJBXDBUG  ANALYSE STANDALONE DUMP ROUTINE
ANEXIT IJBXSDA   SDAID BUFFER FORMATTING ROUTINE
$$
/*
/&
```

Figure 66. Sample Job: Loading the External Routines File via OBJMAINT

Label Information for Info/Analysis

Figure 67 on page 172 shows an example of the DLBL and EXTENT information to submit if you want to use the functions of the Info/Analysis program. These labels should be stored in the system standard label area.

```
* LABELS FOR THE SYSDUMP LIBRARY,
*            THE DUMP MANAGEMENT FILE, AND
*            THE EXTERNAL ROUTINES FILE

   ... ... ...
// DLBL SYSDUMP,'VSE.DUMP.LIBRARY',,VSAM,                                X
              CAT=IJSYSCT,                                               X
              DISP=(OLD,KEEP)
// DLBL BLNDMF,'INFO.ANALYSIS.DUMP.MGNT.FILE',0
// EXTENT SYS016,SYSWK1,1,0,9030,15
// DLBL BLNXTRN,'INFO.ANALYSIS.EXT.RTNS.FILE',2011/365,SD
// EXTENT SYS017,SYSWK1,1,0,9045,15
   ... ... ...
```

Note: The dump library is located in VSAM space if initial installation was performed with z/VSE 5.1 or later.

If you did an FSU from z/VSE 4.3 the following shows the labels for 3380 disks.

```
// DLBL SYSDUMP,'VSE.DUMP.LIBRARY',1999/365,SD
// EXTENT SYS020,SYSWK1,1,0,3150,600
```

Figure 67. Example: File Labels for Dump Processing

Functional Overview

When a dump is created, you can use it to solve a problem by taking actions that range from printing the dump symptoms to analyzing the dump in detail. The actions that you take depend on local procedures for dealing with dumps and your own techniques of dump analysis. Info/Analysis is a tool that can be used to enhance these procedures and techniques.

This section presents the stages of a dump's life cycle from problem occurrence to resolution. The ways in which you can use Info/Analysis at each stage are briefly presented.

When a problem occurs during system operation, the detecting component captures the condition of the system in a dump. The component stores the dump in the dump sublibrary designated for the partition that failed. Sometimes, a system operator may detect a problem and use stand-alone dump or other dumping facilities to create a storage dump. Stand-alone dumps are stored on tape or disk.

In either case, a dump contains a copy of system storage, and a symptom record. The symptom record is a collection of failure-related information gathered by the dumping component when the dump is taken or added later by dump analysis routines.

The symptom record may contain:

- A description of the operating environment at the time the problem occurred.
- Symptoms that provide clues to the problem's origins.
- Free-form text and hexadecimal information that may describe the problem.
- Entries that define the format and location of dump data that may be pertinent to the problem. These entries are used when data is displayed in formatted mode.

For further information about the symptom record, see Appendix A, “Symptom Records Overview,” on page 213.

Chapter 14. Dump Symptoms

The first step in dump analysis is to examine any symptoms that are recorded for the problem. This topic discusses the Dump Symptoms function with which you may display or print the problem symptoms collected by the dumping component at the time of a failure or by subsequent analysis routines. The list of symptoms may indicate a new problem or a duplicate of a previously encountered problem. If sufficient symptoms are provided, they may pinpoint the cause of the failure.

The successful use of the Dump Symptoms function is dependent on the presence of a symptom record in the dump you are processing. The symptom record is created by the dumping component when the dump is taken (see Appendix A, “Symptom Records Overview,” on page 213). Figure 68 on page 173 shows an example of the symptom part of a dump.

```
SYSDUMP.DYN.DR100008

ENVIRONMENT:
    CPU MODEL ....... 2097
    CPU SERIAL ...... 3B0B82
    TIME ............ 12:59:42:00
    DATE ............ 12/10/12
    SYSTEM ID ....... 5686CF906
    RELEASE ......... 5
    FEATURE ......... 1C
    DUMPTYPE ........ SCPREQ
    PROBLEM NUMBER .. ........

REQUIRED SYMPTOMS:
    AB/S2000
    REGS/03818
    REGS/0C990
    MS/0S03I
    RIDS/EYU9XZUT
    OFFS/00000A08
    AB/S0007

OPTIONAL SYMPTOMS (SDB):

OPTIONAL SYMPTOMS (NON-SDB):
    JOB_NAME=EYU9XZUT
    DUMPED_DATA=R1-PARTITION
```

Figure 68. Dump Symptoms Part

Types of Dump Symptoms

The symptoms are organized into the following sections:

- Environment data
- Required symptoms
- Optional symptoms in structured data base (SDB) format
- Optional symptoms in non-SDB format

The symptoms are initially provided by the dumping component. If you subsequently execute analysis routines for the dump, these routines may add symptoms. A plus sign (+) appears before a symptom if it was added by an analysis routine.

The displayed symptoms may pinpoint the cause of the failure. If not, you may compare these symptoms to the symptoms of other locally reported problems. Your installation should set up a procedure whereby a file of problem symptoms is kept and the symptoms can be compared to one another.

If a satisfactory match is found, the problem is considered a local duplicate. If no match is found and the problem is related to an IBM product, a search of known IBM problems would be a logical next step. You may contact IBM service personnel to request this search. If a duplicate set of symptoms is found in either search, a solution may be immediately available or already under investigation. If no match or too many matches occur after a search, additional analysis is necessary. You may perform this analysis using the Dump Viewing function.

Environment

The environment section of the symptom record describes the environment at the time the dump was created. This section is provided by the dumping component. The CPU, operating system, type of dump, and date and time that the dump was taken are identified. Additional items such as release level may be included.

Required Symptoms

Required symptoms are those considered essential for problem analysis. They are provided by the dumping component or by the dump analysis routines.

All of the required symptoms are likely to occur each time the same error occurs. The format of the required symptoms is standardized so that more effective keyword searches for duplicate problems may be conducted.

Each symptom is formatted with a prefix and the specific data connected by a slash. The required symptoms are described under "Symptom Part Description" on page 183.

Optional Symptoms

Optional symptoms may be provided by the dumping component or by the dump analysis routines. These are additional symptoms that apply to the problem and may be present if the problem recurs. Some of these symptoms, for example the component level, are formatted like the required symptoms. There are also free-form symptoms that may be used in problem analysis, but which are not in standardized format.

Chapter 15. Invoking Info/Analysis

You may use Info/Analysis in two ways:

- Line mode - from the operator console
- Reader mode - from the system input device

In either case, you invoke Info/Analysis by submitting a series of job control statements (JCL) followed by control statements that request Info/Analysis functions. All output is routed to the SYSLST device. The output includes the input control statements, the results of processing, and any messages issued by Info/Analysis. If you are working at a console, Info/Analysis also routes messages there. For information on how to print dumps, see "Printing Dump Information" on page 182.

This topic describes:

- How to invoke Info/Analysis.
- Syntax rules for control statements.
- Each Info/Analysis function and the control statements needed to request it.
- How to end Info/Analysis.

To illustrate this information, the topic includes example sequences of control statements.

Note: If the dump with which you are working is too large to be uploaded using Info/Analysis, see "Uploading Large Dumps From a Standalone Dump Tape" on page 18.

Submitting a Job to Invoke Info/Analysis

You invoke Info/Analysis by submitting the necessary JCL followed by control statements that request functions. You may submit the job either in line mode by entering statements on the console, or in reader mode by submitting a job to the system input device.

Info/Analysis requires a program area of at least 300K bytes and a 24-bit partition GETVIS area of at least 600K bytes. Thus, the partition used for the execution of Info/Analysis should have a minimum of 900K bytes.

If user-written exits are to be used when running Info/Analysis, this size has to be adjusted accordingly. Especially the CICS Transaction Server uses such exits; therefore, a size of at least 4M bytes is necessary to analyze a CICS dump.

With JCL, you must specify any nonstandard system device assignments and pre-allocate and assign any files that you require other than the system libraries. A sample of the JCL for invocation is:

```
// JOB   JCL
// ASSGN SYSLST,00E
// ASSGN SYS020,252   Dump library
// ASSGN SYS016,252   Dump management file
// ASSGN SYS017,252   External routines file

// DLBL SYSDUMP,'VSE.DUMP.LIBRARY',,VSAM,                               X
         CAT=IJSYSCT,                                                   X
         DISP=(OLD,KEEP)
// DLBL BLNDMF,’INFO.ANALYSIS.DUMP.MGNT.FILE’,0
// EXTENT SYS016,SYSWK1,1,0,9030,15
// DLBL BLNXTRN,’INFO.ANALYSIS.EXT.RTNS.FILE’,2011/365,SD
// EXTENT SYS017,SYSWK1,1,0,9045,15

// EXEC  INFOANA,SIZE=300K




....            |
....            |  Info/Analysis
....            |  control statements
....            |

/*
/&
```

Figure 69. Sample Job: Invoke Info/Analysis

Standard Info/Analysis Job Stream

Figure 69 on page 176 shows a sample job to invoke the Info/Analysis program (// EXEC INFOANA). Assume that the dump sublibraries in the library SYSDUMP have already been defined and the label information for the SYSDUMP library resides in the standard label area. Information on the SYSDUMP library can be found under “The SYSDUMP Sublibraries” on page 11.

Once the JCL has been processed, you are at the selection level in Info/Analysis. The program reads for your control statements. An end of input (/*) statement marks the end of these statements. To end your job, enter an end of job (/&) statement.

Control Statement Syntax

You operate Info/Analysis by entering control statements. These control statements specify the major functions you wish to perform (Dump Management, Dump Viewing, etc.) and the information necessary to perform each function. This section describes the syntax rules for entering the control statements.

For a description of the syntax diagrams please read “Understanding Syntax Diagrams” on page xxiii.

Entering Control Statements

The rules for entering the control statements that request Info/Analysis functions are:

- Each card or input line may contain only one control statement.
- A control statement may begin in any column.
- Control statements and their operands may be entered in uppercase or lowercase.
- Control statements must be entered in their complete form; no abbreviations are allowed.

- Each word in a control statement must be a contiguous string of characters.
- Some blanks, at least one, must appear between the words in a statement.
- A blank followed by an asterisk (*) signifies the start of a comment. If the first non-blank character in any control statement is an asterisk, the entire statement is treated as a comment.
- Sequence numbers or other characters should not be placed in columns 73 to 80 of Info/Analysis commands if they are not designated as a comment.

If you enter an invalid control statement in reader mode, the remaining control statements in the job are flushed and the session is canceled. The output indicates the erroneous statement. You should correct the statement and resubmit the job. In line mode, invalid input causes Info/Analysis to flush the control statement. You may then reenter the statement correctly.

Common Control Statements

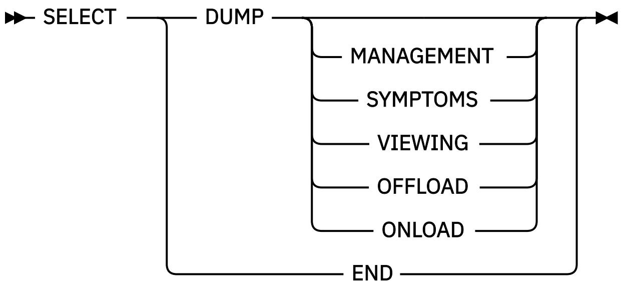
Every function has its own set of control statements. In this topic, the control statement descriptions are presented by function. The same control statement may have different uses within different functions. Therefore statements such as PRINT are described more than once. A summary of the control statements is given under "Control Statement Summary" on page 211.

The following control statements are common to all Info/Analysis functions. They may also be entered at the selection level; that is, when Info/Analysis is initialized and no function is currently selected.

- **SELECT** - invoke a function or end the Info/Analysis session
- **RETURN** - end the current function; return to the selection level
- **DUMP NAME** - specifies the name of the dump to be processed.

SELECT - Specify a Function or End Info/Analysis

To perform any function, you must first select it. Use the SELECT statement to specify the function you wish to perform or to end your Info/Analysis session.



You enter a SELECT statement as follows:

```
SELECT DUMP ONLOAD
```

Once you have specified a SELECT statement, you are operating at the function level. All subsequent statements apply to the selected function until you enter a RETURN statement.

Use SELECT END to conclude your Info/Analysis session. If you end your control statement sequence with an end of input (/*) and end of job (/&) statement without first specifying a SELECT END or RETURN statement, Info/Analysis ends the job as though they had been included.

RETURN - End Current Function

Use the RETURN statement to end the current function. Each function you use must be requested by a SELECT statement and ended by a RETURN statement, before you can select another function.

The RETURN statement has no operands. RETURN brings you back to the selection level where you may select another function or end your session. In the following sequence, RETURN ends the Dump Management function:

```
DUMP NAME SYSDUMP.F6.DF600013
SELECT DUMP MANAGEMENT
PRINT DATA
RETURN
SELECT END
```

You must enter RETURN to complete one function before selecting another. If you end your control statement sequence with an end of input (/*) and end of job (/&) statement without first specifying a RETURN and/or SELECT END statement, Info/Analysis ends the job as if they had been included.

DUMP NAME - Specify or Add Current Dump

Specify the name of the dump you wish to process by entering the DUMP NAME statement. You must specify a dump before you perform any Info/Analysis function except the Dump Management UTILITY or PRINT functions and the HELP function.

►► DUMP NAME *dumpname* ►◄

“dumpname” is a variable representing the dump name. The dump you specify on a dump name statement is considered to be the current dump; that is, all subsequently selected functions process that dump until you enter another dump name or until it has been deleted.

When you specify a dump, Info/Analysis searches the dump management file for the name. If it finds the name, the dump is made current.

If the dump you specify is not identified in the dump management file and does not reside in a dump sublibrary, Info/Analysis adds the dump name and the information “TO BE ONLOADED” to the file. A dump must reside in one of the dump sublibraries before you can act on it using any Info/Analysis function other than Dump Onload.

You may specify the DUMP NAME statement at the selection level. In other words, the DUMP NAME statement may be placed:

- Immediately preceding a SELECT statement, as follows:

```
DUMP NAME SYSDUMP.F3.DF300010
SELECT DUMP MANAGEMENT
PRINT DATA
RETURN
```

- After the SELECT DUMP MANAGEMENT statement and before the next RETURN statement, as follows:

```
SELECT DUMP MANAGEMENT
DUMP NAME SYSDUMP.F3.DF300010
PRINT DATA
RETURN
```

Recommendations (Restrictions) for the Generation of Dump Names

The VSE dump routines generate dump names for ABEND dumps, IDUMPs, and SDUMPS. These system generated dump names have the format 'Dppnnnnn', or 'Sppnnnnn', where the character 'D' identifies dumps of the executing space, and 'S' identifies dumps of associated data spaces. The character combination 'pp' denotes the partition identification, like BG or F7, and 'nnnnn' is a unique decimal number between 00000 and 99999

When Info/Analysis processes a dump via the dump viewing function, it enters additional members into the dump library to save intermediate analysis results. These dumps are named 'Maaaaaaa' or 'Xaaaaaaa', where 'M' or 'X' replace the heading character 'D' or 'S' and aaaaaaa is the unchanged trailing portion of the name of the analysed dump. If you have, for example, a dump named SX400002 in the dump library, then the Infoana program will create the library members MX400002 and XX400002 for its internal use.

You define the names for stand-alone dumps or attention dumps during the onload process. Here are some recommendations which help to build unique dump names.

- Do not use D, H, L, M, N, O, S, X, as the first character of a dump member name. These initial characters are reserved for use by the system.
- Instead use **one** heading letter, for example the letter 'A' as first letter of all dump names within a sublibrary.
- If you use a partition identification within a dump name, do not enter it in positions 2 to 3. This might generate conflicts with system generated names of ABEND dumps, IDUMPS, or SDUMPS.
- D, O, S are used for partition dumps, memory objects and dataspaces.

Dump Management

The Dump Management function enables you to manage your dump data sets by manipulating the contents of the dump management file. This file contains descriptive information about the dumps being managed by Info/Analysis. You can use this information to keep track of the dumps on your system, those you have offloaded, and those you plan to onload.

To initiate Dump Management, use the following statement:

```
▶▶─ SELECT DUMP MANAGEMENT ─▶◀
```

After this selection, specify the desired combination of the following control statements:

UTILITY -
initialize a new or reallocated dump management file to contain the list of dumps managed by Info/Analysis.

DELETE -
erase the current dump from the dump sublibrary, if it resides there, and delete information about the dump from the dump management file.

PRINT DATA -
print the contents of the dump management file.

Info/Analysis responds by searching the dump sublibraries for dumps that are not yet identified in the dump management file. For each of these dumps, if any, the routine adds identifying information to the file.

UTILITY - Initialize Dump Management File

```
▶▶─ UTILITY ─▶◀
```

The UTILITY statement is intended for the system programmer at your installation who has responsibility for Info/Analysis. UTILITY initializes the dump management file at installation time or reinitializes the file when it is subsequently reallocated with more or less space.

The dump management file is allocated during installation of your VSE system. The size of the file is big enough to hold some hundred dump names. (For example, the DMF allocated on a 3380 disk is sufficient for about 1000 dump names.) For performance reasons you should not increase the size of the DMF. Instead it is recommended that you clean-up the dump library from time to time, and delete the dumps which are no more used or offload those dumps to tape which may be used at a later time.

You initialize the dump management file with the following statements:

```
SELECT DUMP MANAGEMENT
UTILITY
RETURN
SELECT END
```

UTILITY sets up the control information in the file for use by the dump management function. The control record indicates the number of dumps currently being managed and the maximum number that will fit.

DELETE - Delete Current Dump

You can erase a dump, and delete its corresponding information in the Info/Analysis dump management file, by using the DELETE statement.

```
►► DELETE ►◄
```

The example in Figure 70 on page 180. shows a job to delete two dumps in one dump management run. Note, that the DUMP NAME statement is valid prior as well as after the SELECT DUMP MANAGEMENT statement.

If you specify DELETE and the current dump does not reside in the library (that is, it has never been onloaded or it has been offloaded), information about the dump is deleted from the dump management file. If a copy of the dump exists on tape, it is your responsibility to dispose that tape.

```
// JOB    DELETE

// ASSGN  SYSLST,00E
 ...

// EXEC INFOANA,SIZE=300K

SELECT DUMP MANAGEMENT                Calls dump management

DUMP NAME SYSDUMP.F5.DR500002         Specifies the dump

DELETE

DUMP NAME SYSDUMP.F5.DF500003
DELETE


RETURN
SELECT END
/*
/&
```

Figure 70. Sample Job: Delete Dumps

Info/Analysis indicates the successful execution of the DELETE function with a message (DUMP dumpname DELETED).

If you want to retain the information about the dump in the dump management file, you can use the SELECT DUMP OFFLOAD operation with the BYPASS operand. Please see Table 16 on page 200, which shows a summary of the OFFLOAD and DELETE functions.

PRINT - Print List of Managed Dumps

Use the PRINT statement to print the contents of the dump management file.

```
►► PRINT DATA ►◄
```

Output from the PRINT statement is a list of the dumps being managed by Info/Analysis. For each dump, one line of information is printed. A sample job is shown in Figure 71 on page 181.

```
// JOB LIST
// ASSGN SYSLST,00E
   ...

// EXEC INFOANA,SIZE=300K

SELECT DUMP MANAGEMENT    |  calls the ‘osq.list
PRINT DATA                |  managed dumps’
RETURN                    |  function

SELECT END

/*
/&
```

Figure 71. Sample Job: List Managed Dumps

An output example of the PRINT DATA function is shown in Figure 72 on page 181.

```
   DUMP NAME          RELATED DUMP ONLINE  DATE/TIME TAKEN   VOLID  DATA SPACE
                                                                       NAME
 SYSDUMP.BG.DBG00000    SBG00006     Y    10/11/21 11:10:00  VOLID0
                        SBG00007
                        SBG00008
 SYSDUMP.BG.DBG00001    --NONE--     Y    10/11/21 11:11:00  VOLID1
 SYSDUMP.BG.DBG00002    --NONE--     Y    10/11/21 11:12:51  VOLID2
 SYSDUMP.BG.DBG00003    SBG00009     Y    10/11/21 11:13:00  VOLID3
                        SBG00010
 SYSDUMP.BG.DBG00004    --NONE--     Y    10/11/21 11:15:53  VOLID4
 SYSDUMP.BG.DBG00005    --NONE--     Y    10/11/21 11:16:54  VOLID5
 SYSDUMP.BG.SBG00006    DBG00000     Y    10/11/21 11:10:10  VOLID6  DATSPAC1
 SYSDUMP.BG.SBG00007    DBG00000     Y    10/11/21 11:10:20  VOLID7  DATSPAC2
 SYSDUMP.BG.SBG00008    DBG00000     Y    10/11/21 11:10:30  VOLID8  DATSPAC3
 SYSDUMP.BG.SBG00009    DBG00003     Y    10/11/21 11:13:20  VOLID9  DATSPAC4
 SYSDUMP.BG.SBG00010    DBG00003     Y    10/11/21 11:13:40  VOLI10  DATSPAC5
```

Figure 72. Example: Dump Management PRINT DATA Output

The column headings represent:

- DUMP NAME - The identifier of the dump.
- RELATED DUMP - Displays the correlation between an ABEND or stand-alone dump and its data space dumps:

  `SYSDUMP.xx.Dxxnnnnn` denotes an ABEND or stand-alone dump. `SYSDUMP.xx.Sxxnnnnn` denotes a data space dump.

  Note that in the column RELATED DUMP, the library and sublibrary of the dump is not displayed, since it is the same as in the corresponding DUMP NAME column.

  For lines containing an ABEND or stand-alone dump in the column DUMP NAME, the column RELATED DUMP contains the names of the data space dumps that were created with the ABEND or stand-alone dump. Vice versa, for lines containing a data space dump in the column DUMP NAME, the second column contains the name of the ABEND or stand-alone dump which was taken together with the data space dump.

  If the entry in the column RELATED DUMP contains --NONE--, the named dump did not access any data space or the data spaces are not indicated in the optional symptoms.

  Information Analysis extracts this information from the optional symptoms of the symptom record.

  When you delete a dump without deleting all related dumps, this overview listing becomes inconsistent, which means that there are no longer lines for all related dumps.
- ONLINE - An indication ("Y", for yes) if the dump is currently stored in the dump sublibrary.
- DATE/TIME TAKEN - The date and time the dump was created or, if the actual date and time are not available, the date and time the dump was identified to Info/Analysis.

"TO BE ONLOADED" indicates that the dump is not in the dump sublibrary (for example a stand-alone dump named to the Info/Analysis management, but not yet onloaded).

- VOLID - The identifier of the tape volume to which the dump has been offloaded or from which the dump has been onloaded, if any. If a dump has been offloaded and then onloaded again, Info/Analysis retains the volume id in the dump management file. Consequently, a dump may be in a dump sublibrary ("Y" in ONLINE field) and still have a VOLID.
- DATA SPACE NAME - The name of the data space (as specified in the DSPSERV macro).

Printing Dump Information

The following sections describe how Info/Analysis is used to print information from the dumps stored on a tape or disk, or in a dump sublibrary.

The contents of a given dump depends on the function which created the dump. But the main form is the same for all dump requesting functions which are able to store dumps on tape, disk, or in a dump sublibrary. The functions described in the subsequent sections can be used for any type of dump stored in a dump sublibrary.

Figure 73 on page 182 gives an overview of the various parts of a dump. Info/Analysis can be used to print these parts selectively.

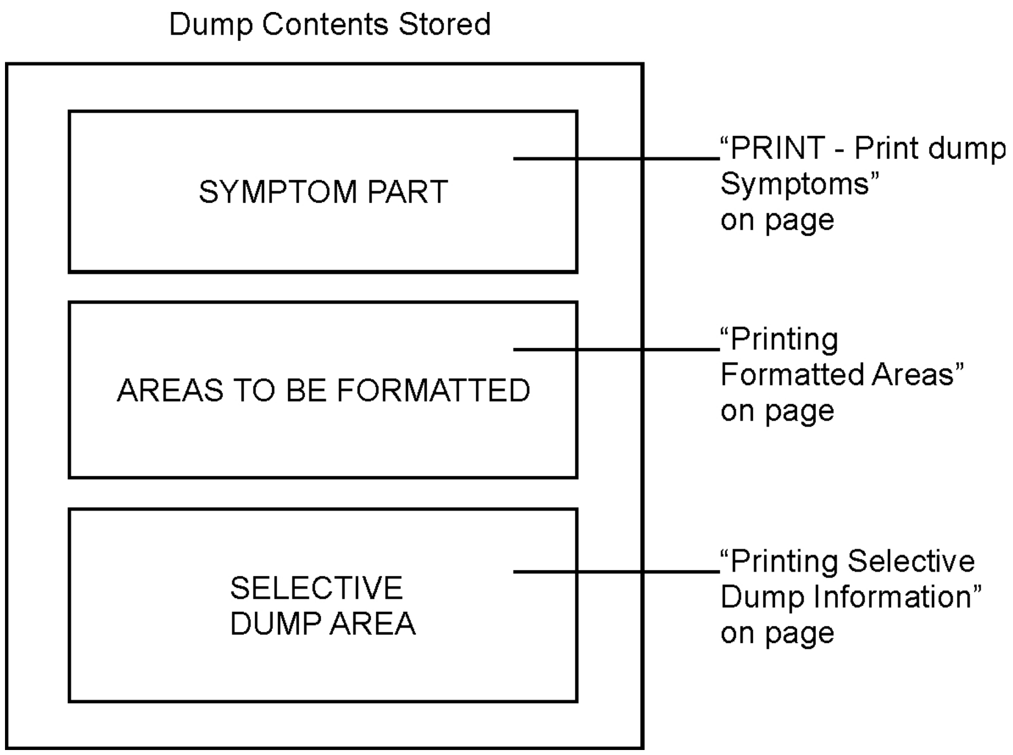

Dump Contents Stored
SYMPTOM PART
AREAS TO BE FORMATTED
SELECTIVE DUMP AREA
“PRINT - Print dump Symptoms” on page
“Printing Formatted Areas” on page
“Printing Selective Dump Information” on page

Figure 73. Overview: Dump Contents

Dump Symptoms

The Dump Symptoms function prints the failure information that is contained in the dump symptom record. You may use this information to identify duplicate problems locally and in an IBM maintenance data base. To initiate Dump Symptoms, enter the following statement:

```
►► SELECT DUMP SYMPTOMS ►◄
```

After making this selection, you may print the symptoms of the current dump using the PRINT statement.

PRINT - Print Dump Symptoms

▸▸— PRINT DATA —▸◂

Use the PRINT statement to print sections 1 through 5 of the symptom record of the current dump. For a description of a symptom record, see Appendix A, “Symptom Records Overview,” on page 213.

Figure 74 on page 183 shows a sample print dump symptoms job.

```
// JOB PRINT
  .
  .
  .
// EXEC INFOANA,SIZE=300K

DUMP NAME SYSDUMP.F3.DF300010        defines the
                                     dump name

SELECT DUMP SYMPTOMS                 calls the
PRINT DATA                           print operation

RETURN

SELECT END
/*
/&
```

Figure 74. Sample Job: Print Dump Symptoms

Symptom Part Example

“Printed Output of the Main Dump File of a Stand-Alone Dump” on page 205 shows an output example of a print dump symptoms operation.

```
ENVIRONMENT:
    CPU MODEL ....... 2097
    CPU SERIAL ...... 000018
    TIME ............ 12:25:48:00
    DATE ............ 12/10/22
    SYSTEM ID ....... 5686CF906
    RELEASE ......... 5
    FEATURE ......... 1C
    DUMPTYPE ........ SADUMP
    PROBLEM NUMBER .. ........

REQUIRED SYMPTOMS:

OPTIONAL SYMPTOMS (SDB):

OPTIONAL SYMPTOMS (NON-SDB):
    DATE_NOT_AVAILABLE
    MACHINE=Z
    MODE=PAGING
    ACTIVE_SPACE_ID=S
    DUMPED_DATA_FROM_SPACE_ID=S
    PMR_ADDRESS_SPACE_ID=00
    DUMPED_DATA=SUPERVISOR+SVA
```

Figure 75. Example: Output of Print Dump Symptoms

Symptom Part Description

ENVIRONMENT

The environment section of the symptom record describes the environment at the time the dump was taken. This data is provided by the dumping system component.

All of the information in the environment section is self-explanatory except the dump type entry, which may be:

SCPREQ
for ABEND dumps

IDUMP
for internal VSE/Advanced Functions dump requests

OPRREQ
for DUMP command dumps

SADUMP
for stand-alone dumps

SDUMP
for SDUMP and SDUMPX macro dumps

REQUIRED SYMPTOMS

Required symptoms are those considered essential to your problem analysis effort:

- The symptoms for ABEND dumps are described below.
- DUMP command dumps only provide the DUMP command itself (as typed in at the console).
- SDUMPs and IDUMPs contain the symptoms that were provided at the respective macro invocation.
- Stand-alone dumps do not contain required symptoms. Details on the dump contents can be obtained by executing the stand-alone dump analysis routine IJBXDBUG, for example.

Each symptom is headed by a prefix; the specific data is connected by a slash. Depending on the cancel code, the following entries might appear in this section:

AB/Sxxyy
ABEND (cancel) code

xx ... first cancel code
yy ... second cancel code or 00 if none exists

```
Example: AB/S0900
```

If a program check occurs, the symptom records contain *two* AB/Sxxyy entries.

- The entry for the ABEND code, *AB*.
- An entry in which *xx* is set to 00 and *yy* contains the program check interrupt code. For example, AB/S0001.

The following symptoms are additional information to the various cancel codes.

ADRS/address
Reflects the absolute address of the instruction following the failing one if the failing address is outside the LTA, SVA, or partition areas (the address is extracted from the PSW).

MS/xxxxx
Message number

```
Example: MS/0V15I
```

OFFS/offset
Offset of the instruction following the failing one relative to the phase start address, if the phase resides in the LTA or in the SVA, or relative to the partition begin address (the address is calculated from the PSW).

OPCS/aaaaaa
aaaaaa Represents either

SVCnn (nn = SVC code) or
CODEmm (mm = program interruption code)

An entry for OPCS is generated only if an illegal SVC or if a program check occurs. If an illegal SVC occurs, the OPCS entry contains the failing SVC code in decimal. If a program check occurs, the entry contains the program interrupt code.

PIDS/comp.id
Component identifier

```
Example: PIDS/5686032V2
```

REGS/xxyyy

- xx is the register number in hex which contains a value, less than and within 4K (K=1024 bytes) of the PSW address at the time of failure. This register contains an address that is close to the point of failure.
- yyy is the difference between the PSW and the register address.

```
Example: REGS/0C14E

0C ...... number of the
          general purpose register X'C'
14E ..... is the difference between the
          PSW address minus the contents
          of register (C).
```

RIDS/Caaaaaaaa
aaaaaaaa stands for either:

- The name of a phase, if the failing instruction is in the SVA or LTA.
- The name of the active job that received the error, if the error occurred in a partition.

```
Example: RIDS/CICSICCF
```

VALU/Caaaaaaaa
aaaaaaaa represents either:

- phase name (for example, combined with AB/S2200 phase not found)
- SYSnnn (for example, together with AB/S2600 SYSnnn not assigned).

<u>REQUIRED SYMPTOMS FOR A MEMORY OBJECT DUMP</u>

A memory object dump starts with its *symptom records*, following by maximum 8K data. The symptom records are:

ADDRESS_SPACE=
address space name

REG=xxyyyyyyyyyyyyyyyy

REG shows the register containing the address that the memory object dump is around.
xx is the register number (hexadecimal).
yyyy… is the content of the register (a 64-bit memory object address).

START_ADDRESS=
start address of the memory object

END_ADDRESS=
end address of the memory object

SHARED=
YES|NO

PAGE_FIXED=
YES|NO

FETCH_PROTECTED=
YES|NO

STORAGE_KEY=
storage key

In addition, the following information is shown *above the dump data:*

FAILED ADDRESS IN MEMORY OBJECT
the 64-bit memory object address contained in the register

DUMP BEGIN
memory object address - '1000'X

DUMP END
memory object address + 'FFF'X

This information is shown only for SYSLST.

OPTIONAL SYMPTOMS (SDB)

SDB stands for structured data base. For information on the symptom record please refer to Appendix A, "Symptom Records Overview," on page 213.

These symptoms (SDB) may be provided by the function which produced the dump or by subsequently executed analysis routines. These additional symptoms apply to the problem and may be present if the problem recurs. They are in SDB format; for example, the component level may be included in this section.

OPTIONAL SYMPTOMS (NON-SDB)

These symptoms (non-SDB) are optionally provided by the dump originating component or by subsequently executed analysis routines. They are free-form symptoms that may be used in problem analysis but do not fit into the SDB format.

Note, that a symptom added by an analysis routine contain a preceding plus sign (+).

Optional (Non-SDB) Symptoms of a Stand-Alone Dump

In addition to the symptoms described above, a symptom record of a stand-alone dump shows the following entries (see also "Printed Output of the Main Dump File of a Stand-Alone Dump" on page 205):

ACTIVE_SPACE_ID
which identifies the address space that was active at the time the dump was taken.

DATA_DUMPED_FROM_SPACE_ID
which indicates the address space from which this dump data file was dumped.

JOB_NAME or DATA_SPACE_NAME

which indicates the job or data space name for this dump file.

DUMPED_DATA
which shows what data is in this dump file.

Optional (Non-SDB) Symptoms of an ABEND Dump

The optional symptoms part of a partition dump shows whether there are any appended data space dumps or not. The symptom gives a list of all data space names which belong to the dumped partition:

```
DATA_SPACES=(aaaa,bb,cccccc)
where
   aaaa,bb,cccccc
is the name of the address space or data space.
```

The symptom record of the data space contains the following symptoms:

```
DATA_SPACE=aaaa
ALET=alet
RELATED_ABEND_DUMP=DF400001
```

Dump Viewing

Using Dump Viewing you may print dump data and analysis summary data. You may also call an analysis routine for execution. You cannot locate particular data or mask data for security purposes. To initiate Dump Viewing, enter the following statement:

```
►►─ SELECT DUMP VIEWING ─►◄
```

After making this selection, you may perform Dump Viewing functions by specifying the following control statements:

- PRINT - print dump data
- CALL - initiate an analysis routine

PRINT - Print Dump Data

Use the PRINT statement to print dump data or analysis summary data. You must specify either the area of the dump you wish to print in hexadecimal mode or that you wish to print all formatted and all analysis summary data. The operands for the PRINT statement are mutually exclusive.

```
►►─ PRINT ─┬──────── from_addr ────────┬─►◄
           ├── from_addr  to_addr ─────┤
           ├──── from_addr  END ───────┤
           ├── from_addr  FOR length ──┤
           └──────── FORMAT ───────────┘
```

The PRINT control statement for Dump Viewing requires either an address range or the FORMAT operand. The results of PRINT with the addr-range operand are printed in traditional hexadecimal format with EBCDIC translation (refer to "Printing Selective Dump Information" on page 188). The FORMAT operand prints the analysis summary if stored during a previous analysis routine run, and formatted dump display information (refer to "Printing Formatted Areas" on page 188). This data includes control blocks, text data, hexadecimal data, and control block linkage descriptors as defined in section 6 of the symptom records. For a description of the symptom records, see Appendix A, "Symptom Records Overview," on page 213.

PRINT

from_addr
'from-addr' marks the beginning of the 8192 byte area to be printed.

from_addr to_addr
'from_addr' marks the beginning of the area to be printed and 'to_addr' marks the end.

from_addr END
'from_addr' marks the beginning of the area to be printed and 'END' indicates that the data up to the high address end of the dumped storage is to be printed.

from_addr FOR length
'from_addr' marks the beginning of the area to be printed and 'length' represents the number of bytes in hexadecimal which are to be printed. For example, if you specify 10 as a length, 16 bytes are printed.

All addresses are 1- to 4-byte hexadecimal values representing valid addresses in the dump. Leading zeros are not required for an address specification.

FORMAT
causes the data to be printed with correlated field names and other identifiers. The data printed is determined by information in section 6 of the symptom records.

For print job examples, see Figure 76 on page 188 and Figure 77 on page 189.

Printing Selective Dump Information

The PRINT statement of the Info/Analysis SELECT DUMP VIEWING function can be used to print dump information selected by addresses.

All addresses and the length setting are 1- to 8-character hexadecimal values representing valid addresses in the dump. Leading zeros are not required for an address specification. The specification of

```
PRINT 0 END
```

for example, would cause the whole dump data to be printed.

The example in Figure 76 on page 188 shows the:

- Definition of the dump SYSDUMP.F3.DF300010 to be processed.
- Definition of the selective print operation for dump data, beginning at the address X'302000' and ending at dump end.

```
// JOB PRINT
 .
 .
 .
// EXEC INFOANA,SIZE=300K
                            |
DUMP NAME SYSDUMP.F3.DF300010 |  defines the
                            |  dump name
                            |
                            |
SELECT DUMP VIEWING         |  defines the
PRINT 302000 END            |  area to be
                            |  printed
RETURN                      |
SELECT END                  |
/*
/&
```

Figure 76. Sample Job: Print Selected Dump Areas

Printing Formatted Areas

The printed output of a dump is called formatted if selected system information is extracted and printed separately.

The example in Figure 77 on page 189 shows how to print the dump SYSDUMP.F3.DF300010 formatted on the device at SYSLST.

```
// JOB PRINT
  .
  .
  .
// EXEC INFOANA,SIZE=300K          |
                                   |
DUMP NAME SYSDUMP.F3.DF300010      |  defines the
                                   |  dump name
                                   |
SELECT DUMP VIEWING                |  calls the
PRINT FORMAT                       |  PRINT FORMAT
                                   |  function
                                   |
RETURN                             |
SELECT END                         |
/*
/&
```

Figure 77. Sample Job: Print a Dump in Formatted Form

CALL - Initiate Analysis Routine

Use the CALL statement to invoke an analysis routine.

▶▶— CALL *routine_name* —▶◀

The routine_name is required on a CALL statement and must be the name of an executable routine. These routines may be provided by system components or by your location. It is the responsibility of your location to maintain an external routines file containing the names of executable routines. Consult your system programmer for the names of these routines.

A call job example is given in Figure 78 on page 190. It shows a job which

- Selects the dump SYSDUMP.BG.ADUMP10;
- Calls the analysis routine IJBXCSMG.
- Calls the analysis routine IJBXDBUG.
- Calls the analysis routine IJBXSDA.

Analysis routines can be used to analyze the stored dumps. For example, the routine IJBXDBUG (shipped as part of VSE/Advanced Functions) analyzes the output of stand-alone dumps and adds the analysis information to the dump sublibrary.

For a description of the analysis routines available with VSE, see "The Stand-Alone Dump Analysis Routine IJBXCSMG" on page 190, "The Stand-Alone Dump Analysis Routine IJBXDBUG" on page 190 and "The Stand-Alone Dump Analysis Routine IJBXSDA" on page 197.

```
// JOB ANALYZE

// ASSGN  SYSLST,00E
 ...

// EXEC INFOANA,SIZE=300K

DUMP NAME SYSDUMP.BG.ADUMP10   |  defines the
                               |  dump name
                               |
SELECT DUMP VIEWING            |  defines the
CALL IJBXCSMG                  |  analysis
CALL IJBXDBUG                  |  routine
CALL IJBXSDA                   |  call
                               |
RETURN                         |
SELECT END                     |
/*
/&
```

Figure 78. Sample Job: Call the Analysis Routines IJBXCSMG, IJBXDBUG and IJBXSDA

The Stand-Alone Dump Analysis Routine IJBXCSMG

IJBXCSMG is an exit routine of Info/Analysis which processes the console events by redisplaying about the last most recent 20 messages and inputs including the timestamp and the console name where the source is coming from. IJBXCSMG prints the last console entries on SYSLST.

Activating the routine

The routine must be contained in the Info/Analysis external routines file before you can call the routine.

Start execution of the analysis routine with the Info/Analysis statements:

```
SELECT DUMP VIEWING
CALL IJBXCSMG
```

Figure 78 on page 190 illustrates the activation of an analysis routine.

The Stand-Alone Dump Analysis Routine IJBXDBUG

The analysis routine IJBXDBUG analyses the output of a stand-alone dump onloaded to a dump sublibrary.

When IJBXDBUG receives control from Info/Analysis, it requests portions of dump data using Info/Analysis dump access routines. During analysis of the dump data, information from the dump is extracted and written back to the dump sublibrary for further Info/Analysis operations.

Activating the Routine

The routine name IJBXDBUG must be contained in the Info/Analysis external routines file before you can call the routine.

The statements

```
SELECT DUMP VIEWING
CALL IJBXDBUG
```

of Info/Analysis start execution of the analysis routine.

Figure 78 on page 190 illustrates the activation of an analysis routine.

Output of the Routine

The output of the analysis routine may contain the following:

- General information.
- Specific information.

While general analysis information is provided in every dump, specific analysis information concerns only particular error situations. The output of the analysis routine is stored in the dump sublibrary. It can be printed together with formatted dump areas with the operation:

```
DUMP NAME ...........
SELECT DUMP VIEWING
PRINT FORMAT
```

General Analysis Information

The general analysis information, which is provided for each dump, is not dependent on certain error conditions.

Header entry

This contains data as follows:

- Service level identifier
- Supervisor ID
- Supervisor name
- Date the dump was taken
- Dump type
- System status
- Current task
- Owner of LTA and transient name (if active)

Address Validation

When an address has been located or calculated during the analysis process of the dump data, it is validated. IJBXDBUG checks whether the address is

- Within the range of high and low limits of the affected areas:
  - Supervisor
  - Partition
  - SVA.

When an address is found to be invalid, the address and information about the expected contents of the address are added to the dump. An address validation entry might look like this:

```
INVALID ADDRESS FF0002 ENCOUNTERED DURING ANALYSIS.
ADDRESS OF: PUB TABLE FROM BG COMREG
```

Specific Analysis Information

The data which is selected, analyzed, and finally stored in the dump sublibrary depends on the error situation. The following error conditions can be recognized by the IJBXDBUG routine:

- Hard Waits
  - WAITFFA
  - WAITFFB
  - WAITFFF
  - WAITFF9
  - WAITFFE
  - WAITFD0
  - Other Hard Waits
- Soft Waits or System Running (Loop)

Hard Wait Dump Entries

For all dumps that indicate a hard wait, the routine supplies

- Wait state code
- Hard wait reason code
- General purpose registers at the time the failure occurred
- Access registers at the time the failure occurred

Besides this information, the following data, depending on the specific hard wait code, is provided:

For *WAITFFA, WAITFFB, or WAITFFF*

- Type of program check
- Program check address
- Instruction at the program check address
- Overwritten instruction information (if applicable)
- Name of transient which program-checked and the displacement within the transient (if applicable)
- Transient areas checked are LTA, PTA, DOC, and RTA
- Name of the SVA phase which program-checked and the displacement within the phase (if applicable)
- Registers at the time of failure
- 64 bytes of data pointed to by each register

For *WAITFF9 or WAITFFE*

- Last device to which a sense was issued
- Sense data address
- Sense data
- Registers at the time of failure
- 64 bytes of data pointed to by each register.

For *WAITFD0*

- IPL cancel code
- IPL cancel reason code
- Registers at the time of the failure
- 64 bytes of data pointed to by each register

For *other Hard Waits*

- Registers at the time of the failure
- 64 bytes of data pointed to by each register

Soft Waits or System Running

On all dumps indicating a soft wait or a system running condition, the status of all active devices (non-telecommunication) and active tasks is supplied. Information is provided for active devices, excluding local telecommunication devices. A device is active if:

- It is flagged busy in the PUB table
- It has a channel queue entry queued to the PUB table.

A task is active if it is not unbatched, stopped, or flagged not active in the TIB (Task Information Block). Tasks of VSE/POWER are classified as not active if VSE/POWER has flagged the task as waiting for work. The following information is provided:

Device Status Information

- Device address
- Device type
- Task-id of first channel queue entry
- I/O request status (I/O started or not started)
- Reason I/O not started (CSW stored, intervention, etc.)
- CSW from channel queue entry if interrupt has been presented
- A list of additional tasks with I/O queued for this device
- The device, that last presented an interrupt to the system

Task Status Information

- Task name
- Serviced task name
- Main task name for subtask
- Status
- What the task is waiting for
- Task information (LTA active, ICCF pseudo partition, EOT active, etc.)
- Subsystems running in the partition

Output Examples

Examples of IJBXDBUG output are given:

- For a hard wait X'FFF' - See Figure 79 on page 194.
- For a system loop - See Figure 80 on page 195.

```
SYSTEM STATUS:  HARD WAIT                           HARD WAIT CODE:  FFF
CURRENT TASK:  AR TASK


HARD WAIT REASON CODE:  24 - PROGRAM CHECK IN SUPERVISOR

PROGRAM OLD PSW INDICATES 31 BIT ADDRESSING MODE.

PROGRAM CHECK TYPE:  0010 SEGMENT TRANSLATION EXCEPTION
ADDRESS OF PROGRAM CHECK:  A0D1E4E8
PROGRAM CHECK INSTRUCTION:  58600014

SYSTEM STATUS:  HARD WAIT
CURRENT TASK:  AR TASK




DEVICE ANALYSIS FOR ACTIVE NON TP DEVICES ONLY:

DEV  TYPE  TSK  I/O REQUEST STATUS AND INFORMATION
---  ----  ---  -------------------------------------------------------------
009  3277  N/A  LAST I/O INTERRUPT WAS FROM THIS DEVICE
(NO BUSY DEVICES AND NO DEVICES WITH I/O QUEUED)



TASK ANALYSIS FOR ACTIVE TASKS ONLY:

TASK NAME     STATUS    TASK INFORMATION
------------  --------  -----------------------------------------------------
CMT TASK      WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
DSP TASK      WAITING   ON DISPATCHER SERVICE
CST TASK      WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
HCF TASK      WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0005D608
FCP TASK      WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
AR TASK       READY TO   RUN
F1 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0027C040
                        SUB SYSTEMS IN THIS PARTITION:  POWER
                        LIBRARIAN SERVICE ACTIVE
F2 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SUB SYSTEMS IN THIS PARTITION:  ICCF CICS IUI
                        LIBRARIAN SERVICE ACTIVE
SYSDUMP.BG.SAHW0001                                                                                PAGE 000011

                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
F3 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0028FBB8
                        SUB SYSTEMS IN THIS PARTITION:  VTAM
                        LIBRARIAN SERVICE ACTIVE
FB MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T0099 FB SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00503CA0
T009A FB SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T009B F1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00509DF8
                        LIBRARIAN SERVICE ACTIVE
T009C F3 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T009D F3 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T009E F3 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 002985D4
T009F F3 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00555698
T00A0 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00A1 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00A2 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00A3 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00A4 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00A5 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00A6 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00A7 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00A83538
```

Figure 79. Example: WAITFFF Analysis Report

```
SYSTEM STATUS:  RUNNING                             PSW:  04040000 00000000 00000000 0020D6E4
CURRENT TASK:  AR TASK

AREA POINTED TO BY PSW:  SVA                        PHASE NAME:  $IJBAR



DEVICE ANALYSIS FOR ACTIVE NON TP DEVICES ONLY:

DEV  TYPE  TSK  I/O REQUEST STATUS AND INFORMATION
---  ----  ---  ---------------------------------------------------------------
F00  (E0)  T00  I/O STARTED, AWAITING INTERRUPT
                DEVICE END POSTING REQUIRED
F02  (E0)  T00  I/O STARTED, AWAITING INTERRUPT
                DEVICE END POSTING REQUIRED
F04  (E0)  T00  I/O STARTED, AWAITING INTERRUPT
                DEVICE END POSTING REQUIRED
F06  (E0)  T00  I/O STARTED, AWAITING INTERRUPT
                DEVICE END POSTING REQUIRED
F08  (E0)  T00  I/O STARTED, AWAITING INTERRUPT
                DEVICE END POSTING REQUIRED
F0A  (E0)  T00  I/O STARTED, AWAITING INTERRUPT
                DEVICE END POSTING REQUIRED



TASK ANALYSIS FOR ACTIVE TASKS ONLY:

TASK NAME     STATUS    TASK INFORMATION
------------  --------  -------------------------------------------------------
CMT TASK      WAITING   FOR LOG. TRANSIENT AREA
                        SVC RETRY INDICATOR ON      SVC: 02 (HEX)
DSP TASK      WAITING   ON DISPATCHER SERVICE
CST TASK      WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
HCF TASK      WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0005CD18
FCP TASK      WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
T1F TASK      WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
AR TASK       READY TO   RUN
BG MAIN TASK  WAITING   FOR VSE/POWER SPOOLING
F1 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0028C040
                        SUB SYSTEMS IN THIS PARTITION:  POWER
                        LIBRARIAN SERVICE ACTIVE
F2 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SUB SYSTEMS IN THIS PARTITION:  ICCF CICS IUI
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
F3 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 002B4BB8
                        SUB SYSTEMS IN THIS PARTITION:  VTAM
                        LIBRARIAN SERVICE ACTIVE
F4 MAIN TASK  WAITING   FOR PARTITION GETVIS
                        TASK IS ACTIVE IN LTA
                        TASK IS LTA OWNER
                        TERMINATOR ACTIVE FOR TASK
                        EOT CLEANUP IN PROCESS
                        LIBRARIAN SERVICE ACTIVE
F5 MAIN TASK  WAITING   FOR LOG. TRANSIENT AREA
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 02 (HEX)
F9 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SUB SYSTEMS IN THIS PARTITION:  CICS IUI
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
FB MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
S1 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
I1 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        JOB CONTROL ACTIVE IN THIS PARTITION
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
R1 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 006EE0EC
                        LIBRARIAN SERVICE ACTIVE
R2 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
```

Figure 80. Example: System Loop Analysis Report (1 of 3)

```
U1 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
R3 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        JOB CONTROL ACTIVE IN THIS PARTITION
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T1 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SUB SYSTEMS IN THIS PARTITION:  CICS IUI
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
O1 MAIN TASK  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SUB SYSTEMS IN THIS PARTITION:  CICS IUI
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T0099 FB SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00503BA8
T009A FB SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T009B FB SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T009C F1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0050A210
T009D F1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00501838
                        LIBRARIAN SERVICE ACTIVE
T009E F3 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T009F F3 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00A0 F3 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 002BB5D4
T00A1 F3 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0054BA98
T00A2 F3 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00543070
T00A3 F3 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 005430D0
T00A4 I1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00A5 R2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 005FBCC4
T00A6 R2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0054D04C
T00A8 I1 SUB  WAITING   FOR LOG. TRANSIENT AREA
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 02 (HEX)
T00A9 I1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 005C2358
T00AA F9 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00AB F9 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00AC R2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 005FB20C
T00AD F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00AE F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00AF F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00B0 F2 SUB  WAITING   FOR LOG. TRANSIENT AREA
                        SVC RETRY INDICATOR ON      SVC: 02 (HEX)
T00B1 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00B2 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00B3 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00A83538
T00B4 F2 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00B5 S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00547E10
                        SYSTEM CODE ACTIVE (LIBR)
T00B6 S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0052BD90
T00B7 S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0052BD88
T00B8 U1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00B9 S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SYSTEM CODE ACTIVE (LIBR)
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00BA S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 005CA990
                        SYSTEM CODE ACTIVE (LIBR)
T00BB S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 005B8C10
T00BC S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 005B8C08
T00BD F9 SUB  WAITING   FOR LOG. TRANSIENT AREA
                        SVC RETRY INDICATOR ON      SVC: 02 (HEX)
T00BE T1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
```

Figure 81. Example: System Loop Analysis Report (2 of 3)

```
T00BF F9 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00C0 T1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00C1 T1 SUB  WAITING   FOR LOG. TRANSIENT AREA
                        SVC RETRY INDICATOR ON      SVC: 02 (HEX)
T00C2 F9 SUB  WAITING   FOR LOG. TRANSIENT AREA
                        SVC RETRY INDICATOR ON      SVC: 02 (HEX)
T00C3 F9 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00C4 S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00644990
                        SYSTEM CODE ACTIVE (LIBR)
T00C5 S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 005BFE90
T00C6 S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 005BFE88
T00C7 S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 006B0990
                        SYSTEM CODE ACTIVE (LIBR)
T00C8 S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00648F10
T00C9 S1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00648F08
T00CA 01 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        CONSOLE REQUEST PENDING
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00CB 01 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00CC 01 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00CD 01 SUB  WAITING   FOR LOG. TRANSIENT AREA
                        SVC RETRY INDICATOR ON      SVC: 02 (HEX)
T00CE 01 SUB  WAITING   FOR LOG. TRANSIENT AREA
                        SVC RETRY INDICATOR ON      SVC: 02 (HEX)
T00CF 01 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00D0 01 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00D1 T1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
T00D3 I1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0054609C
T00D5 I1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 0052012C
                        LIBRARIAN SERVICE ACTIVE
T00D6 01 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00D7 F9 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00D8 I1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        SVC RETRY INDICATOR ON      SVC: 1D (HEX)
T00DB T1 SUB  WAITING   FOR I/O, ECB OR TECB        CCB/ECB ADDRESS: 00000000
                        LIBRARIAN SERVICE ACTIVE
                        SVC RETRY INDICATOR ON      SVC: 84 (HEX)
```

Figure 82. Example: System Loop Analysis Report (3 of 3)

The Stand-Alone Dump Analysis Routine IJBXSDA

The analysis routine IJBXSDA formats the contents of the SDAID buffer in a stand-alone dump, if SDAID was active when the dump was taken. IJBXSDA prints the trace entries on SYSLST.

Activating the Routine

The routine name IJBXSDA must be contained in the Info/Analysis external routines file before you can call the routine.

Start execution of the analysis routine with the Info/Analysis statements:

```
SELECT DUMP VIEWING
CALL IJBXSDA
```

Figure 78 on page 190 illustrates the activation of an analysis routine.

Dump Offload

Dump Offload places a dump file that resides on the dump library onto tape for later retrieval. You may choose whether or not to maintain the copy that is on the dump library; the default is to erase the dump. To initiate Dump Offload, use the following statement:

```
►► SELECT DUMP OFFLOAD ►◄
```

After making this selection, you may offload the current dump by specifying any of the following control statements that are necessary:

- VOLID - specify the output volume and the logical unit number

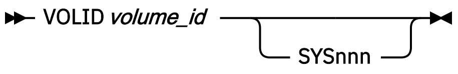
- BYPASS - skip the write-to-tape operation
- ERASE NO - does not delete the library copy of the dump

Dump Offload is valuable if you need to increase the available online space. Dump Offload does not remove the information about the dump from the dump management file.

An example of a dump Offload job is given in Figure 83 on page 199.

VOLID - Specify Output Volume

Use the VOLID statement to specify the identifier of the output tape.



The volume_id is a 6-character alphameric value that is added to the entry for the current dump contained in the dump management file. The VOLID statement is required if the dump you are offloading has not been previously onloaded or offloaded.

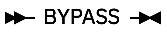
For subsequent offloads of the same dump, Info/Analysis can retrieve the volume_id from the dump management file. To override the saved volume, use the VOLID statement. The most recent VOLID is the one saved in the dump management file.

The logical unit number (SYSnnn) can be assigned to a physical tape address via the job control ‘ASSGN’ statement. If you do not define a logical unit number allocation is done automatically by the system. The first available unit will be allocated and a message issued for the tape to be mounted on this unit.

BYPASS - Skip Offload

Use the BYPASS statement to free the library space used by the dump without writing the dump to tape. BYPASS is allowed only if a valid offloaded copy of the dump exists; that is, if both of the following conditions are met:

- The current dump has been previously offloaded, but is still in the library, and
- The current dump has not been modified by an analysis routine since it was last offloaded.



When BYPASS is processed, Info/Analysis checks for the above conditions. If they are not met, the offload function is not performed.

The DUMP Offload BYPASS statement and the Dump Management DELETE statement differ in the following ways:

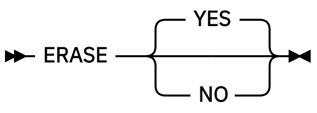
- BYPASS checks for a copy of the dump on tape. DELETE does not.
- DELETE removes references to the dump from the dump management file. BYPASS does not.

Thus, use Dump Management with DELETE only if you no longer need a dump. Use Dump Offload with BYPASS if you wish to remove a dump from the dump library but want to maintain a copy on tape and keep information about the dump in the dump management file.

The BYPASS and ERASE NO statements are contradictory and thus mutually exclusive.

ERASE - Delete or Retain Library Copy of Dump

The ERASE statement specifies whether or not Info/Analysis should delete the dump library copy of the dump when doing an offload.



If you want to maintain a copy of the dump on the dump library as well as on tape, specify ERASE NO. ERASE YES is the default. Therefore, if you specify ERASE, ERASE YES, or do not specify the ERASE control statement during Dump Offload, Info/Analysis erases the dump from the dump library after a copy is offloaded to tape.

By specifying ERASE NO, you can offload more than one copy of the dump. For example, you may offload a copy of the dump, then run analysis routines, then offload the modified copy. The ERASE statement updates the dump management file with offload information.

BYPASS and ERASE NO are contradictory and thus mutually exclusive statements.

Offloading a Dump to Tape

Figure 83 on page 199 shows how to offload a dump to a tape, erase the copy in the dump sublibrary and update the Info/Analysis dump management file.

```
// JOB  OFFLOAD
                                   |
// ASSGN SYS009,280                |  Assigns tape to receive
                                   |  offloaded dump
// MTC REW,280                     |  Rewind tape to loadpoint

// ASSGN SYSLST,00E
 ...

// EXEC INFOANA,SIZE=300K
                                   |
DUMP NAME SYSDUMP.F4.DF400003      |  Defines the dump name
                                   |
                                   |
SELECT DUMP OFFLOAD                |  Calls OFFLOAD
                                   |
                                   |
VOLID T07111 SYS009                |  Defines the tape
                                   |  volume and optionally
                                   |  the logical unit
RETURN                             |
SELECT END                         |

/*
/&
```

Figure 83. Offloading a Dump to Tape

If the tape unit is not defined via the ‘ASSGN’ and ‘VOLID’ statements during the OFFLOAD process, Info/Analysis searches for a free tape drive and issues a volume mount request message (MOUNT VOLUME volumename ON UNIT xxx ....).

Info/Analysis indicates the successful execution of the OFFLOAD function with a message (DUMP dumpname OFFLOADED...).

SELECT DUMP OFFLOAD versus SELECT DUMP MANAGEMENT DELETE

In comparison to the SELECT DUMP OFFLOAD the SELECT DUMP MANAGEMENT DELETE operation described in “Dump Management” on page 179 has the following functions:

- Erases the dump from the dump sublibrary;
- Erases the information about the dump from the dump management file.

Table 16 on page 200 summarizes the functions of the SELECT DUMP OFFLOAD and the SELECT DUMP MANAGEMENT DELETE operation and shows the differences between these two operations.

Table 16. Summary: SELECT DUMP OFFLOAD and DELETE Operation

Dump written to tape	**Information kept in Dump Management File**	**Dump erased from Sublibrary**	**Info/Analysis Function**
YES	YES	YES	OFFLOAD without additional operands
YES	YES	NO	OFFLOAD with ERASE NO specified
NO	YES	YES	OFFLOAD with BYPASS specified
NO	NO	YES	DELETE

Dump Onload

Dump Onload copies dumps which reside on tape or disk into the dump library so that they can be further processed by Info/Analysis. To initiate Dump Onload, use the following statement:

```
►► SELECT DUMP ONLOAD ►◄
```

After making this selection, you may onload the current dump by entering the VOLID and FILE control statements if necessary.

An example of a dump Onload job is given in Figure 84 on page 202.

VOLID - Specify Input Volume

Onloading a Dump from Tape

Use the VOLID statement to specify the volume identifier and (optional) the logical unit number of the tape on which the current dump resides.

```
►► VOLID volume_id ─┬──────────┬─►◄
                    └─ SYSnnn ─┘
```

The volume_id is a 6-character alphameric value that is added to the entry for the current dump contained in the dump management file. The VOLID statement is required if you are onloading a dump for the first time.

For subsequent offloads and onloads of the dump, Info/Analysis retrieves the volume_id from the dump management file. To override the saved value, use the VOLID statement. The most recent VOLID is the one saved in the dump management file.

The logical unit number (SYSnnn) can be assigned to a physical tape address via the job control 'ASSGN' statement. If you do not define a logical unit number allocation is done automatically by the system. The first available unit will be allocated and a message issued for the tape to be mounted on this unit.

Specification of SYSnnn is highly recommended to prevent difficulties in mixed tape environments.

Onloading a Stand-Alone Dump from Disk

Use the VOLID statement with the DISK operand to specify the disk device on which the current dump resides.

```
►► VOLID DISK SYSnnn ►◄
```

DISK indicates that the stand-alone dump is to be copied from the disk device with the logical unit number SYSnnn. Note that SYSnnn is mandatory when onloading a dump from disk.

FILE - Specify Dump File on Multiple-Dump Device

If the tape or disk you are using contains more than one dump file, use the FILE statement to specify the specific dump file you want to onload. In this way, you may onload more than one dump from a tape or disk during a session. Keep in mind that you must leave Dump Onload and specify another dump name before onloading the next file.

```
►► FILE file_ number ──┬────────┬──►◄
                       └─ LAST ─┘
```

The default for the FILE statement is "1" if the file statement is omitted. Therefore, if you are onloading a dump from a single file tape/disk or if you are onloading the first file from a multiple-file tape/disk, you need not specify the FILE statement. Also, for the main dump file of a stand-alone dump, the FILE statement need not be specified.

The file number must designate an existing file on the input device. This sequence number is used for searching by Dump Onload when the input file is opened.

When multiple dumps are onloaded during an Info/Analysis session, their file numbers do not have to be in ascending order.

The LAST parameter indicates that this is the last file to be onloaded from the current volume. Specifying LAST de-allocates the device from Info/Analysis.

Loading a Dump into a Dump Sublibrary

Dumps can be stored on tape or disk as output of the following functions:

- DUMP command (tape)
- Standalone Dump Program (tape or disk)
- Info/Analysis Offload operation (tape)

Before you can process these dumps, they have to be onloaded into a dump sublibrary.

The example in Figure 84 on page 202 shows such an onload job; it stores the dump with the name SYSDUMP.BG.DMPLO3 in the dump sublibrary assigned to the BG partition.

```
// JOB ONLOAD

// ASSGN SYS009,281                1. Assigns dump tape to SYS009

// ASSGN SYSLST,00E
 ...

// EXEC INFOANA,SIZE=300K

DUMP NAME SYSDUMP.BG.DMPL03         2. Defines dump name

SELECT DUMP ONLOAD                  3. Calls ONLOAD


VOLID T03111 SYS009                 4. Specifies volume and
                                       optional logical unit

FILE 2 LAST                         5. Specifies second file


RETURN
SELECT END
/*
/&
```

Figure 84. Sample Job: Onload a Dump from Tape into a Dump Sublibrary

The sample job in Figure 84 on page 202:

1. Assumes that the tape containing the dump is mounted on the tape drive at address 281. Assigns programmer logical unit SYS009 to this drive.
2. Specifies the dump for processing, by name. The dump sublibrary is determined by the dump name.
3. Calls the Info/Analysis DUMP ONLOAD function.
4. Specifies the tape volume on which the dump resides.

   The volume name is provided in the list of managed dumps if the dump in question has been offloaded before. See "PRINT - Print List of Managed Dumps" on page 180, for the list function.

   If the dump on tape you want to onload has not been offloaded before, the volume id is an identifier of your own choosing. It is used to identify the volume in subsequent dump operations.
5. Defines the file sequence number 2 with the LAST operand in the FILE statement (the dump resides on a multifile tape volume, sequence number 1 is the default value).

When the dump to be onloaded is on a disk extent, make sure that the DLBL/EXTENT statements for the IJSYSDU file are available (see "Dump Program File and Dump Data Set" on page 29). During the ONLOAD process Info/Analysis searches for a free tape drive if the tape unit is not defined via the 'ASSGN' and 'VOLID' statements and issues a volume mount request message (MOUNT VOLUME volumename ON UNIT xxx ....).

Info/Analysis indicates the successful execution of the ONLOAD function with a message (DUMP dumpname ONLOADED).

Printing a Dump Stored on Tape or Disk

The subsequent sections describe methods to print dumps that were written to tape or disk, like the stand-alone dump, or to tape, like the DUMP command dump.

The **Info/Analysis program** can be used to format and print these dumps after they have been onloaded to a dump sublibrary. The steps which have to be performed to print the dump are described under "Processing and Printing a Dump with Info/Analysis" on page 203.

The **DOSVSDMP utility program** can be used to write dumps in unformatted form directly from the dump device to the printer. This utility program **must** be used to print dumps produced in response to the attention routine command:

```
DUMP BUFFER,cuu
```

These dumps must not be onloaded to the dump sublibrary.

DOSVSDMP can also be used to print other dumps which are too large for the dump sublibrary and therefore cannot be handled by the Info/Analysis program.

Figure 85 on page 203 shows the different steps which you have to perform depending on the print method you choose.

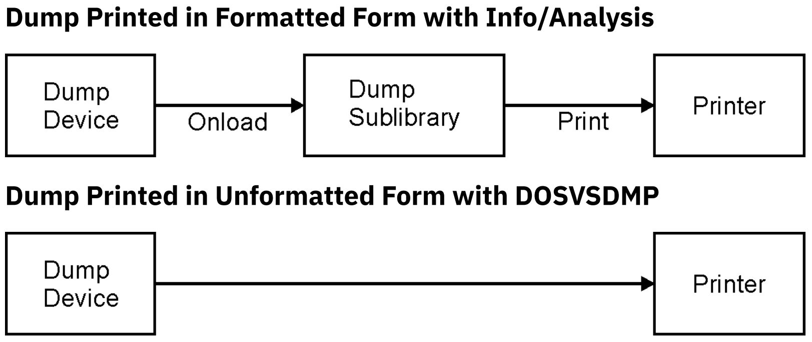

Dump Printed in Formatted Form with Info/Analysis
Dump Device
Onload
Dump Sublibrary
Print
Printer
Dump Printed in Unformatted Form with DOSVSDMP
Dump Device
Printer

Figure 85. Overview: Print a Dump

The following sections describe how to print a dump with Info/Analysis and also how to use the DOSVSDMP utility for the same purpose. Sample jobs are included and the output is explained.

Processing and Printing a Dump with Info/Analysis

This section describes the steps which have to be performed in order to get a printed output of the stand-alone dump or the DUMP command dump. The dump has to be in one of the dump sublibraries before Info/Analysis can be used to print information from it. Therefore the first operation to be done is to onload the dump into one of the dump sublibraries.

Onloading the Dump into the Dump Sublibrary

The following describes in detail the steps which have to be performed.

1. Define the name of the dump.

   The name of the dump determines the dump sublibrary. For example:

   ```
   SYSDUMP.BG.DUMPSA2
   ```

   defines the sublibrary BG as the target library for the subsequent onload process.
2. Select the Info/Analysis Onload function.
3. Use the VOLID statement to define the tape or disk volume on which the dump resides.
4. If the dump resides on tape, mount the dump tape on the device which Info/Analysis requests during processing, or which you defined with the VOLID statement.
5. When the dump has been stored in the desired dump sublibrary, Info/Analysis can print the dump.

Printing a Stand-Alone Dump with Info/Analysis

Note: You cannot use Info/Analysis to print 64-bit *memory objects*. For further information about memory objects, refer to the topic "System Organization and Concepts" in the *z/VSE Planning*, SC34-2681.

Following are two examples of typical stand-alone dump print jobs and their output.

Sample Job to Print the Main Dump File of a Stand-Alone Dump

The example shown in Figure 86 on page 204 defines a job which

- Onloads a stand-alone dump with the user defined name SYSDUMP.BG.ADUMPSA2.
- Prints the following on the device assigned to SYSLST:

    The symptoms of the dump.
    The formatted areas of the dump.

```
// JOB    PRINT

// ASSGN SYSLST,00E
 ...

// EXEC INFOANA,SIZE=300

DUMP NAME SYSDUMP.BG.ADUMPSA2      |  1. Defines the dump name
                                   |
SELECT DUMP ONLOAD                 |  2. Selects ONLOAD
                                   |
VOLID T03111 SYS009                |  3. Defines the input device
                                   |
RETURN                             |
                                   |
SELECT DUMP SYMPTOMS               |  4. Calls the DUMP SYMPTOMS
PRINT DATA                         |     function
                                   |
RETURN                             |
                                   |
SELECT DUMP VIEWING                |  5. Calls the DUMP VIEWING
PRINT FORMAT                       |     function
                                   |
RETURN                             |
                                   |
SELECT END                         |
/*
/&
```

Figure 86. Sample Job: Print a Stand-Alone Dump (Main Dump File)

You can also print the dump data using

```
SELECT DUMP VIEWING
PRINT 0 END
```

which does no formatting of the dump. See also "Printing Selective Dump Information" on page 188.

Sample Job to Print an Additional File of a Stand-Alone Dump

The example shown in Figure 87 on page 205 defines a job which

- Onloads a stand-alone dump with the user defined name SYSDUMP.BG.ADUMPSA4.
- Prints the following on the device assigned to SYSLST:

  The symptoms of the dump.
  The formatted areas of the dump.

```
// JOB    PRINT

// ASSGN SYSLST,00E
 ...

// EXEC INFOANA,SIZE=300

DUMP NAME SYSDUMP.BG.ADUMPSA2    |  1. Defines the dump name
                                 |
SELECT DUMP ONLOAD               |  2. Selects ONLOAD
                                 |
VOLID T03111 SYS009              |  3. Defines the input device
                                 |
RETURN                           |
                                 |
SELECT DUMP SYMPTOMS             |  4. Calls the DUMP SYMPTOMS
PRINT DATA                       |     function
                                 |
RETURN                           |
                                 |
SELECT DUMP VIEWING              |  5. Calls the DUMP VIEWING
PRINT FORMAT                     |     function
                                 |
RETURN                           |
                                 |
SELECT END                       |
/*
/&
```

Figure 87. Sample Job: Print a Stand-Alone Dump (Additional Dump File)

You can also print the dump data using

```
SELECT DUMP VIEWING
PRINT 0 END
```

which does no formatting of the dump. See also "Printing Selective Dump Information" on page 188.

Printed Output of the Main Dump File of a Stand-Alone Dump

Figure 88 on page 206 shows the output (symptom part) of the sample job given in Figure 86 on page 204. A list of the formatted dump areas which are printed are given in Figure 89 on page 206 and Figure 90 on page 207.

```
ENVIRONMENT:
    CPU MODEL ....... 2097
    CPU SERIAL ...... 100190
    TIME ............ 13:48:04:00
    DATE ............ 13/03/13
    SYSTEM ID ....... 5686CF906
    RELEASE ......... 5
    FEATURE ......... 1C
    DUMPTYPE ........ SADUMP
    PROBLEM NUMBER .. ........

REQUIRED SYMPTOMS:

OPTIONAL SYMPTOMS (SDB):

OPTIONAL SYMPTOMS (NON-SDB):
    DATE_NOT_AVAILABLE
    MACHINE=Z                             Shows the hardware type.
    MODE=PAGING                           Indicates paging.
    ACTIVE_SPACE_ID=S                     Shows which address space was active
                                          at the time the dump was taken.
    DUMPED_DATA_FROM_SPACE_ID=S           Shows which address space this dump
    PMR_ADDRESS_SPACE_ID=00               data file was dumped from.
    DUMPED_DATA=SUPERVISOR+SVA            Shows what data is in this dump file.
```

Figure 88. Sample: Symptom Part of the Stand-Alone Dump Output (Main Dump File)

The following stand-alone dump information is selected and printed.

```
Formatted Areas of the Stand-Alone Dump:
```

PSW
Program Status Word (at time of failure)

AREGS
Access Registers

FREGS
Floating Point Registers

GREGS
General Purpose Registers

CREGS
Control Registers

MESSAGE
Error messages and the last 200 messages from the Hard-Copy File

```
Hexadecimally Displayed Areas of the Stand-Alone Dump Output:
```

LOWCORE
Low address storage

SYSCOM
System Communication Region

UNATTCB
Re-IPL control block (previous Re-IPL invocation)

UNATTCBN
Hard Wait information (last Re-IPL invocation)

SMCOM
Storage Management Communication Area

CLIM
Class/System Limits Control Block

Figure 89. Summary of the DUMP VIEWING, PRINT FORMAT Operation Output (1 of 2)

PCB	Partition Control Block
SCB	Space Control Block
PMRAS	Page Manager Address Space
ASTE	Address Space Number Second Table Entry
PASNAL	Primary Address Space Number Access List
COMREG	Partition Communication Region
PIBTAB	Partition Information Block
PIB2TAB	Partition Information Block Extension
LUBTAB	Logical Unit Block
PUBTAB	Physical Unit Block
PUB2TAB	Physical Unit Block Extension
ERBLOC	Error Recovery Block
CHQTAB	Channel Queue Table
CHNTAB	Channel Control Table
TIBATAB	Task Information Block Address Table
TIB	Task Information Block
TCB	Task Control Block
SAVAREA	Partition Save Areas
ACCREGS	Access Registers
TDSE	Task's Data Space Extension
DUCT	Dispatchable Unit Control Table
DUAL	Dispatchable Unit Access List
DSCB-SCB	Data Space Space Control Block
LPT	Library Pointer Table
LDT	Library Definition Table
SDT	Sublibrary Definition Table
EDT	Extent Definition Table
DDT	Device Definition Table
SDBUFFER	SDAID Buffer

Figure 90. Summary of the DUMP VIEWING, PRINT FORMAT Operation Output (2 of 2)

DUMP Command Dump Printed with Info/Analysis

This section gives an example of how to print a DUMP command dump that is on tape; it also shows the output of the sample job.

Sample Job to Print a DUMP Command Dump

The example shown in Figure 91 on page 208 defines a job to print the dump symptoms and the formatted dump areas of a DUMP command dump named SYSDUMP.BG.ADUMPC02 on the device assigned to SYSLST.

An example of the output of this job is shown under Figure 92 on page 209 and Figure 93 on page 209.

```
// JOB PRINT

// ASSGN SYS009,280

// ASSGN SYSLST,00E
 ...

// EXEC INFOANA,SIZE=300K
                               |
DUMP NAME SYSDUMP.BG.ADUMPC02  |  1. Defines the dump name
                               |
SELECT DUMP ONLOAD             |  2. Selects ONLOAD
                               |
VOLID T03111 SYS009            |  3. Defines the input
                               |     tape volume
RETURN                         |
                               |
SELECT DUMP SYMPTOMS           |  4. Defines the
PRINT DATA                     |     print operation
                               |
RETURN                         |
                               |
SELECT DUMP VIEWING            |  5. Defines the
PRINT FORMAT                   |     print operation
                               |
RETURN                         |

SELECT END

/*
/&
```

Figure 91. Sample Job: Print the Output of a DUMP Command

Output of the DUMP Command Dump Printed by Info/Analysis

The output of the DUMP command dump printed by Info/Analysis consists of two parts:

- The dump symptom part (example shown in Figure 92 on page 209)
- The formatted dump areas (summary of the areas shown in Figure 93 on page 209).

Figure 91 on page 208 shows the job with which these two dump parts can be produced.

Output of the DUMP SYMPTOMS, PRINT DATA Operation

```
ENVIRONMENT:
    CPU MODEL ....... 2097
    CPU SERIAL ...... 100190
    TIME ............ 13:48:04:00
    DATE ............ 13/03/13
    SYSTEM ID ....... 5686CF906
    RELEASE ......... 5
    FEATURE ......... 1C
    DUMPTYPE ........ OPRREQ
    PROBLEM NUMBER .. ........


REQUIRED SYMPTOMS:
DUMP                    which is the command
50000/65000             that requested the dump
```

Figure 92. Sample: Symptom Part of the DUMP Command Dump

Output of the DUMP VIEWING, PRINT FORMAT Operation

Figure 93 on page 209 gives a list of the dump areas printed.

PSW
Program status word at time of failure

AREGS
Access registers

GREGS
General purpose registers at time of failure

FREGS
Floating point registers

LOADLS
Phase load list of the partition

Figure 93. Summary: Areas to be Printed with DUMP VIEWING, PRINT FORMAT

Ending the Info/Analysis Job

You end an Info/Analysis job by submitting the SELECT END statement while you are at the selection level. The selection level is the point in a sequence after a RETURN statement and before a function is selected. If you wish to end your session and are at the function level, enter RETURN followed by SELECT END. The function level is the point in a sequence after a function is selected and before a RETURN statement is entered.

SELECT END should be followed by an end-of-input statement (/*) and an end-of-job statement (/&). If you enter an end-of-input or end-of-job statement at any point in the sequence, the job is canceled at that point. Any valid control statement sequences preceding the end-of-input or end-of-job statement are performed as specified.

Control Statement Sequence Examples

The following are examples of batch execution sequences. Each example describes a possible sequence of functions and presents the control statements to perform those functions.

Each function and the statement that selects that function are labeled with the same letter so that you may make comparisons easily. The example in Figure 94 on page 210 contains the following operations:

1. Select the Dump Management function and request the printing of the list of managed dumps.
2. On the selection level, specify SYSDUMP.F6.DF600007 as the current dump.

3. Use the Dump Symptoms function to print the dump symptoms that are contained in the symptom record.
4. Use the Dump Viewing function to print selective areas of the dump. The assumed areas are written in the comments on each statement.
5. Use Dump Offload to offload SYSDUMP.F6.DF600007 to the tape with VOLID T02512.
6. End your Info/Analysis session.

```
1.  SELECT DUMP MANAGEMENT
      PRINT DATA
      RETURN
2.  DUMP NAME SYSDUMP.F6.DF600007
3.  SELECT DUMP SYMPTOMS
      PRINT DATA
      RETURN
4.  SELECT DUMP VIEWING
      PRINT 0 20880          * PRINT SUPERVISOR DATA
      PRINT C80000 END       * PRINT TO END OF STORAGE
      PRINT FORMAT           * PRINT ALL FORMATTED DATA
      RETURN
5.  SELECT DUMP OFFLOAD
      VOLID T02512
      RETURN
6.  SELECT END
```

Figure 94. Control Statement Sequence Example

The example in Figure 95 on page 211 contains the following operations:

1. On the selection level, specify SYSDUMP.BG.ADUMPSA6 as the current dump.
2. Use Dump Onload to load the current dump (file 3 on tape T300U1) into the dump sublibrary so that you can work with it.
3. Use Dump Viewing to call routine IJBXDBUG to analyze the stand-alone dump. Results of the routine are printed together with all formatted data.
4. On the selection level, specify SYSDUMP.BG.ADUMPSA2 as the current dump.
5. Use Dump Offload to offload SYSDUMP.BG.ADUMPSA2, specifying the output volume and choosing to bypass the write operation because a valid copy of the dump already exists on tape. (The information concerning this dump in the dump management file will be kept.)
6. On the selection level, specify SYSDUMP.BG.ADUMPSA7 as the current dump.
7. Use Dump Onload to load the current dump (file 5 on tape T300U1) into a dump sublibrary, specifying LAST because it is the last dump to be onloaded from the tape.
8. End your Info/Analysis session.

```
1. DUMP NAME SYSDUMP.BG.ADUMPSA6

2. SELECT DUMP ONLOAD
     VOLID T300U1
     FILE 003
     RETURN

3. SELECT DUMP VIEWING
     CALL IJBXDBUG
     PRINT FORMAT
     RETURN

4. DUMP NAME SYSDUMP.BG.ADUMPSA2

5. SELECT DUMP OFFLOAD
     VOLID T03417
     BYPASS
     RETURN

6. DUMP NAME SYSDUMP.BG.ADUMPSA7

7. SELECT DUMP ONLOAD
     VOLID T300U1
     FILE 5 LAST
     RETURN

8. SELECT END
```

Figure 95. Control Statement Sequence Example

Control Statement Summary

This section contains a summary of the control statements for Info/Analysis. The statements are presented in alphabetical order. The “Valid Functions” column represents the functions during which the control statement may be entered as follows:

```
M   - Dump Management
S   - Dump Symptoms
V   - Dump Viewing
OF  - Dump Offload
ON  - Dump Onload
SEL - Selection level
T   - Tutorial
```

Summary of Control Statements for Info/Analysis

		Valid Functions					
Control Statement	**Description**	**M**	**S**	**V**	**OF**	**ON**	**SEL**
BYPASS	skip offload				X		
CALL routine	call analysis routine			X			
DELETE	delete dump	X					
DUMP NAME dumpname	specify dump	X					X
ERASE YES|NO	delete/retain system copy of dump				X		
FILE number LAST	specify dump file					X	
PRINT addr-range|FORMAT|DATA	print dump data or formatted dump			X			
	print dump management file	X	X				
RETURN	end function	X	X	X	X	X	X
SELECT function|END	select function						X

Summary of Control Statements for Info/Analysis *(continued)*							
		Valid Functions					
Control Statement	**Description**	**M**	**S**	**V**	**OF**	**ON**	**SEL**
UTILITY	initialize dump management file	X					
VOLID volume-id SYSnnn	specify input or output tape				X	X	
VOLID DISK SYSnnn	specify input disk					X	

In the PRINT command, addr-range can be:

- from-addr
- from-addr to-addr
- from-addr END
- from-addr FOR length

In the SELECT command, function can be:

- DUMP MANAGEMENT
- DUMP SYMPTOMS
- DUMP VIEWING
- DUMP OFFLOAD
- DUMP ONLOAD

Appendix A. Symptom Records Overview

The symptom records contain a collection of failure-related symptoms in a standard format. At the time of problem detection, the failing component creates the symptom records. Subsequently, analysis routines may be run to collect additional symptoms and may add them to the symptom records. The ultimate goal of the symptom records is to reduce the amount of time necessary to analyze a dump.

Symptom Records Structure

The symptom records have six sections. Figure 96 on page 213 shows an overview of the symptom records contents and the information used in a dump.

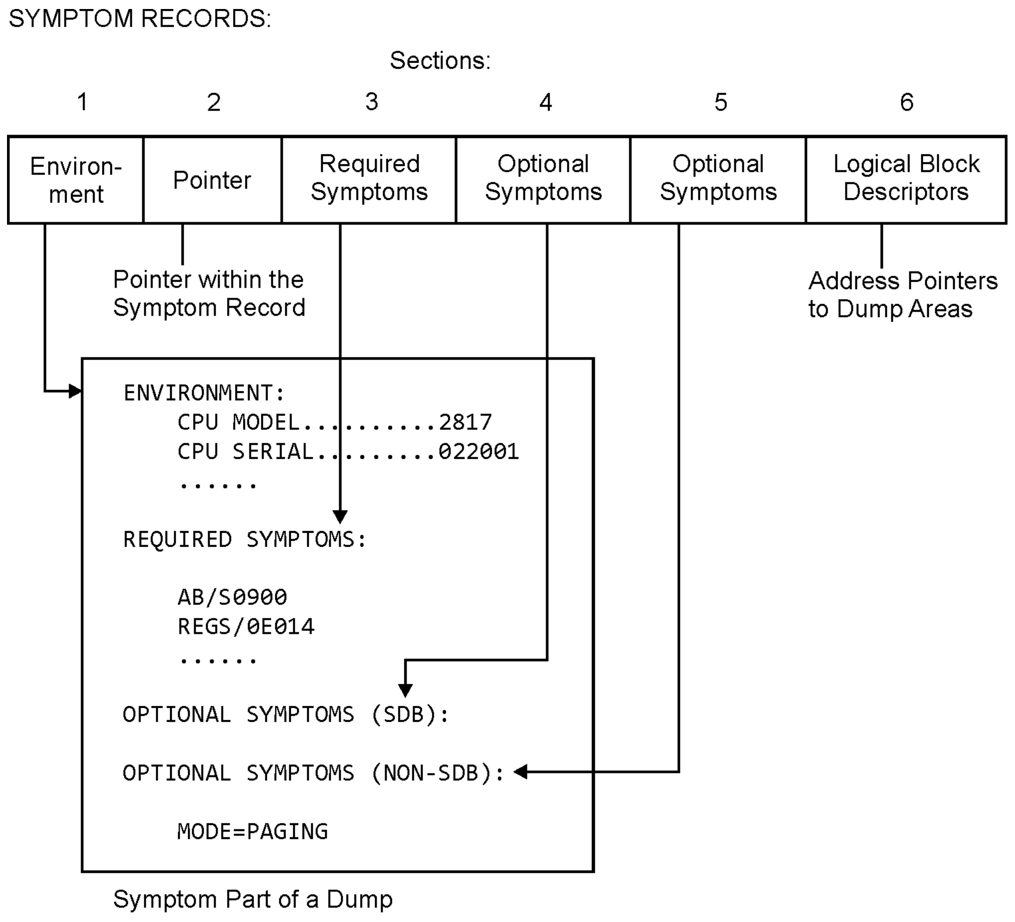


Figure 96. Symptom Records

The sections of the symptom records are:

1. The environment section, which describes the operating system at the time the problem is detected.
2. The pointer section, which describes the offsets of each of the other sections from the beginning of the record and the length of each section. This data is used by the analysis tool and is not accessible to you.
3. The required symptoms section, which contains symptoms considered essential to your dump analysis tasks. Required symptoms are in the structured data base (SDB) format.
4. The optional symptoms (SDB) section, which contains additional symptoms in the SDB format.
5. The optional symptoms (non-SDB) section, which contains symptoms that do not conform to the SDB format but provide failure-related information.

6. The control block section, which contains descriptions and locations of control blocks that are necessary for problem analysis. This section also contains text and hexadecimal entries that may be related to the problem and descriptions of control block chains and arrays.

The contents of Sections 1, 3, 4, and 5 is displayed when you select Dump Symptoms. This information can help you determine the nature of the problem and where it occurred.

The contents of Section 6 is used by Info/Analysis to display dump information when you select Dump Viewing. The control block data presented via the Section 6 entries can help you determine why the problem occurred. See the discussion of Section 6 below.

Note: The structured data base format is used to standardize problem data so that searches for duplicate problems in the data base of existing problems used by customer engineering are accurate.

Symptom Record Creation

The symptom records are built when:

- A system component detects an error that may or may not result in a dump. The dumping component builds the symptom records, completing the required symptoms section and as much optional data as possible. The dumping component then calls a system dumping routine that fills out the environment section, merges it with the rest of the symptom records, and possibly takes a dump.
- A stand-alone dump is taken. The environment section is completed by the dump program.

Section 6

Section 6 acts as a table of contents for dump data. You can use it to locate certain control blocks without having to manually follow pointers through the dump. The component which originates the dump is responsible for providing information about the control blocks that are pertinent to the error.

The section provides for example:

- Names and locations of dumped control block storage.
- Descriptions of conditions present at the time of the dump.
- Hexadecimal data that may not be contained in the dumped storage (such as registers).
- Algorithms for control block relationships.
- The names of control block fields.

The location of Section 6 entries is in separate records of the dump that are classified as symptom record extensions.

At dump time, the failing component may designate the control blocks that are suspected of being in error or are necessary for problem determination. Ideally, the dump includes only the storage in use when the error occurred. Host system storage that provides pointers to the component address space or partition may not be needed. This practice reduces the volume of dump output.

The failing component may include descriptions of the related control blocks in Section 6. Keep in mind that the data for the control blocks is within the main body of the dump. The information in Section 6 describes the addressing method, the content, the format, and the chaining structure of these related dump areas.

Each of these descriptive entries in Section 6 is called a locating block descriptor (LBD). LBDs come in a variety of forms to describe the structure and relationships of control blocks. Each LBD consists of a header segment and, optionally, a variable segment. The header identifies the data being described by the LBD by providing a name, its length, and usually, its location in storage. The optional portion may be:

- One or more extensions
- A formatting descriptor
- A linkage descriptor

By including a variety of LBDs, Section 6 becomes a table of contents for dump data. Through the Dump Viewing function of Info/Analysis, you can use Section 6 to analyze the dump.

Locators

If a header portion of an LBD provides a control block address, a specific instance of a control block has been identified. An LBD may still be a locator if the address is not provided. This would be the case for a linkage descriptor, for example.

A locator may be simple or complex. A simple locator names a control block and defines its length, address qualifications, and other pertinent information. A simple locator is appropriate if there is one occurrence of a particular control block in a dump or in a linkage. A complex locator consists of a header portion resembling a simple locator and one or more extensions that:

- Provide a way to find all occurrences of the control block that is defined in the header.
- Associate additional data with the control block defined in the header.

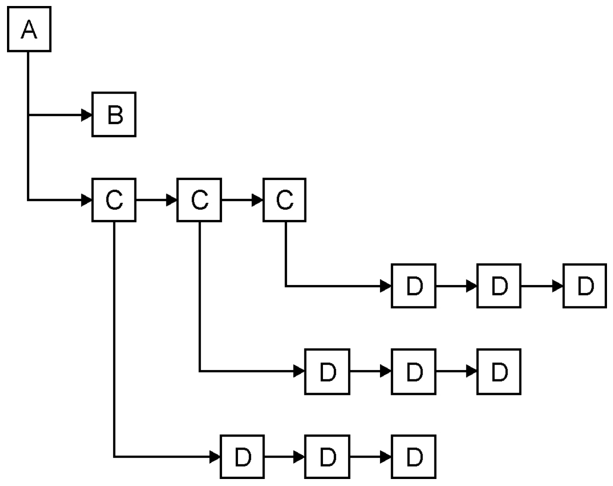
There are five kinds of locator extensions:

- Chain extension - describes a string of all occurrences of one type of control block in a dump. If you select a linkage descriptor while viewing the analysis summary main display, and the locator for a particular type of control block has a chain extension, Info/Analysis uses the locator and its chain extension to display the chain of control blocks of that type.
- Array Extension - describes contiguous occurrences of one type of control block in a dump. If you select a linkage descriptor while viewing the analysis summary main display, and the locator for a particular type of control block has an array extension, Info/Analysis uses the locator and its array extension to display the array of control blocks of that type.
- Text Extension - contains character data added to the dump by the dumping component. The text is associated with the name defined in the locator's header. You may view text for which the header merely provides a name by selecting this text entry from the analysis summary main display. Alternatively, the header may describe a control block with which the text is associated. If you select a locator while viewing the analysis summary LOCATORS display, and the locator by that name has a text extension, the lines of text are displayed along with the control block data.
- Hexadecimal Extension - contains hexadecimal information such as register contents or storage data areas that may pertain to the software problem that caused the dump. This data is collected and associated with the name defined in the header portion of the locator. You may view hexadecimal data for which the header merely provides a name by selecting the hexadecimal entry from the analysis summary main display.
- Keyfield Extension - contains the location of a particular field (such as a completion code) that is significant or pertinent to the associated control block identified in the header portion of the locator. This field may be in (or apply to) the control block. According to the status of this field, you can decide whether or not to continue to examine the control block. The keyfield is included in an analysis summary linkage or LOCATORS display.

Linkage Descriptors

A linkage descriptor is a special type of extension to a header that defines the relationships for a set of control blocks. That is, the control block named in the linkage is used to locate one or more control blocks that are logically related.

Linkage descriptors are used in conjunction with locators. Together, they enable Info/Analysis to display an ordered list of control blocks. For example, a component might create a linkage descriptor that defines control block A pointing to B and a chain of Cs, and each C pointing to a chain of Ds. When you select the linkage entry for that component while viewing the analysis summary main display of Dump Viewing, the displayed data depicts the following situation:



Info/Analysis shows the linkages by indenting control block names. To create the display, Info/Analysis uses both locators with their extensions, if any, and a linkage descriptor.

Formatting Descriptors

A formatting descriptor is a special type of extension to a header that defines a set of simple formatting instructions for a control block. The header portion of the formatting descriptor cannot be used to locate a control block in the dump.

A formatting descriptor for a control block maps out some or all of the fields of the control block named in the associated locator, their offsets from the beginning of the control block, and their lengths. When you select a control block for display while in the formatted mode of dump display, Info/Analysis displays the contents of each field, one or more per line. Depending on the options you have set, the field labels and offsets may appear with the data. To create such a display, Info/Analysis uses both a locator and a formatting descriptor.

Appendix B. Other Diagnosis Tools

This appendix describes various commands and facilities to process information relevant for problem analysis.

ACTION: Print Linkage Editor Map

```
►► ACTION MAP ►◄
```

To obtain a linkage editor map for a program, specify, for the program's linkage editor run, the linkage editor control statement ACTION MAP.

Figure 97 on page 218 shows a sample output from this routine.

Linkage Editor Map Warning Messages

The following messages may be included in the map output, except when NOMAP was specified in the ACTION statement for the linkage editor run.

Root structure overlaid by succeeding phase

When this message appears, "OVEROOT" is printed to the left of the phase name that overlays the root phase.

Possible invalid entry point duplication in input

An entry label appeared at least twice in the input. At the second (or later) appearance it was not possible to validate it as being a true duplication. The most common reason for this message is sub-modular structure with (source) entry labels defined before the CSECT in which the entry point appears.

Invalid transfer label on end or entry statement ignored

An overriding transfer label in the entry statement was not defined within the first phase, or a transfer label was not defined in an end statement in its own module.

Control sections of zero length in input

The COBOL, FORTRAN, RPG*, and PL/1 (D) compilers do not supply all of the information required by the linkage editor in the ESD records. Specifically, the control section length is provided in the end record. If a control section defined in the ESD information has a length of zero, it normally indicates that the length is to appear in the end record. It is possible to generate zero-length control sections through Assembler. Such a condition produces this message. This is not an invalid condition if it is not the last control section that is of zero length. If the last control section is of zero length, the length is implied to be in the end record and, if not present, causes an error condition.

Unresolved external references

These labels indicate external references that cannot be matched with a corresponding entry point. ESD items from unused control sections may also cause this message.

```
// JOB LINK MODABCD                                          DATE 10/30/10,CLOCK 14/12/11
// LIBDEF PHASE,CATALOG=IJSYSRS.SYSLIB
// LIBDEF OBJ,SEARCH=(IJSYSRS.GL)
// OPTION CATAL
  INCLUDE MODLNK
 PHASE PHASE2,*,SVA
  MODE AMODE(31),RMODE(24)
  ACTION ERRLMT(1000),XXXX
  INCLUDE MODC
 ENTRY CS1B
// EXEC LNKEDT,PARM='AMODE=ANY'
JOB LINK     10/30/10  5686-066-06-15C-0  LINKAGE EDITOR DIAGNOSTIC OF INPUT
INVOCATION PARAMETERS:  AMODE=ANY RMODE=24◄──────────────── (A)
ACTION TAKEN MAP◄──────────────────────────────────────────── (B)
 INCLUDE MODLNK◄───────────────────────────────────────────── (C)
            **MODULE MODLNK  10-10-30 11.30 10-10-30 12.36 INCLUDED FROM  IJSYSRS .GL    VOLID=ESA132
-----------------------------------------------------------
PHASE PHASE1,*  ◄──────────────────────────────────────────── (C,G)
-----------------------------------------------------------
INCLUDE MODA    ◄──────────────────────────────────────────── (C)       0000001
            **MODULE MODA    10-10-30 07.51 10-10-30 12.14 INCLUDED FROM IJSYSRS  .GL    VOLID=EDA132
INCLUDE MODB    ◄──────────────────────────────────────────── (C)       0000001
            **MODULE MODB    10-10-30 07.59 10-10-30 13.56 INCLUDED FROM IJSYSRS  .GL    VOLID=ESA132
----------------------------------------------------------
PHASE PHASE2,*,SVA  ◄──────────────────────────────────────── (C,G)
----------------------------------------------------------
 MODE AMODE(31),RMODE(24)
ACTION TAKEN ERRLMT=1000
21351 INVALID OPERAND IN ACTION STATEMENT ◄────────────────── (K)
         ACTION ERRLMT(1000),XXXX ┌──── (D)    ┌──── (E)                 ┌──── (F)
 INCLUDE MODC ◄────────────── (C) ▼            ▼                         ▼
            **MODULE MODC    10-10-30 08.05 10-10-30 09.20 INCLUDED FROM IJSYSRS .GL     VOLID=ESA132
            **MODULE MODD    10-10-30 08.06 10-10-30 12.13 AUTOLNKD FROM IJSYSRS .GL     VOLID=ESA132
                                                              ▲
                                                              └────────────── (H)
ENTRY CS1B
10/30/10 COMMON AREAS:                    NAME    LOADED AT    LENGTH
                                          COM1A   2AB078            8─┐
                                          COM1B   2AB080          280 │>(I)
                                                  2AB300           20 │
                                          COM2B   2AB320           68─┘
10/30/10 PSEUDOREGISTERS:                 NAME    ORIGIN       LENGTH
                                          PSEU1        0           19─┐
                                          PSEU2       20           78 │>(J)
         TOTAL LENGTH OF PSEUDOREGISTERS:                          98─┘
```

(A) Possible invocation parameters on the PARM field of the EXEC LNKEDT statement are: MSHP, AMODE, RMODE.

(B) Option MAP is default if SYSLST is assigned.

(C) Listing of control statements as submitted to linkage editor. (From Job Control or an included module.)

(D) Date and time the module has been cataloged the first time.

(E) Date and time of last update.

(F) Sublibrary from where the module is included or SYSLNK.

(G) Phase statement. This statement defines the phase name, the load address (for example * to indicate relocatable) and for example, whether the phase has to be SVA eligible or the AUTOLINK feature has to be deactivated. The named phase is composed of the subsequent included modules.

(H) Module auto-linked, based on unresolved external reference 'MODD' in Phase 'PHASE2'.

(I) List of named and unnamed Common Control Sections with name, load address, and length.

(J) List of Pseudo Registers (External Dummy Sections) with name, origin (displacement within PR pool), and length. The total (cumulative) length indicates the amount of storage to be allocated during execution.

(K) Error message from invalid ACTION statement.

Figure 97. Sample: Linkage Editor Output (ACTION MAP) (Part 1 of 2)

```
         (A)    (B)         (C)          (D)   (E)     (F)     (G)       (H)     (I) (J)  (K)         (L)

10/30/10 PHASE  XFR-AD  LOCORE HICORE  CSECT/  LOADED  RELOC.  PARTIT.  PHASE   TAKEN    AMODE/RMODE
                                       ENTRY   AT      FACTOR  OFFSET   OFFSET  FROM
         ------------------------------------------------------------------------------------------------------
         PHASE1 2AB3A8  2AB388 2AB3C3                                                *P  ANY   24  RELOCATABLE
         ------------------------------------------------------------------------------------------------------
                                        CS1A   2AB388  2AB388  000310   000000  MODA     24    24
                                       *LAB1A  2AB388
                                       +LAB1B  2AB38C
                                        CS1M   2AB398  2AB388  000320   000010  MODA     24    24
                                       +LAB1M  2AB398
                                       +LAB1N  2AB399
                                       +LAB1P  2AB39A
                                        CS1N   2AB3A0  2AB388  000328   000018  MODA     24    24
                                       *LAB1Q  2AB3A0
                                       +LAB1R  2AB3A1
                                       *LAB1S  2AB3A2
                                        CS1B   2AB2A8  2AB3A8  000330   000020  MODB     24    24
                                       *LAB1C  2AB3AC
         ------------------------------------------------------------------------------------------------------
         PHASE2 2AB3C8  2AB3C8 2AB3F3                                                *M  31    24  SVA ELIGIBLE
         ------------------------------------------------------------------------------------------------------
                                        CS2A   2AB3C8  2AB3C8  000350   000000  MODC     24    24
                                       *LAB2A  2AB3C8
                                       +LAB2B  2AB3CC
                                        MODD   2AB3D8  2AB3D8  000360   000010  MODD     24    24
                                       *LAB3A  2AB3D8
                                       +LAB3B  2AB3DC
UNRESOLVED EXTERNAL REFERENCES                EXTRN    LAB1T
UNRESOLVED ADCON AT OFFSET 002AB3C0                                >(M)
001 UNRESOLVED ADDRESS CONSTANTS
PHASE(S) CATALOGED INTO SUBLIBRARY IJSYSRS.SYSLIB  VOLID= ESA132
1S55I LAST RETURN CODE WAS 0004
EOJ LINK   MAX.RETURN CODE=0002                             DATE 10/30/10,CLOCK 14/12/16,DURATION   00/00/04
```

(A) Name of each phase.

(B) Address where the phase is transferred to.

(C) Lowest and highest virtual storage location of the phase.

(D) Labels of all CSECTs which establish the phase in ascending order followed by the CSECT's entry labels. + indicates an entry label, which was referenced and * indicates an entry label which was not referenced.

(E) CSECT load address / ENTRY address.

(F) Difference between the start of virtual storage and the assembled CSECT start address.

(G) Offset from the partition begin plus save area length to the CSECTs start location.

(H) Offset from the phase begin to the CSECTs start location.

(I) Name of the module from which the CSECT is taken.

(J) Contains either *M or *P (or blank) of the phase:
*P indicates that the AMODE/RMODE assigned from ESD data is overridden by values from the PARM field of the EXEC LNKEDT control statement.
*M indicates that the AMODE/RMODE assigned from ESD data or from the PARM field is overridden by values from the MODE control statement.
The field is left blank, if neither the PARM field nor a MODE statement specifies AMODE/RMODE.

(K) Contains AMODE and RMODE of phases and CSECTs.

(L) Indicates loading characteristics of a phase.

(M) Warning messages related to unresolved external references.

Figure 98. Sample: Linkage Editor Output (ACTION MAP) (Part 2 of 2)

DITTO: Dump a Disk or Tape

▶▶— DITTO —▶◀

IBM Data Interfile Transfer, Testing and Operations/ESA for VSE (DITTO/ESA for VSE), an IBM program product, is a useful tool for the recovery of data that may have become inaccessible by VSE programs.

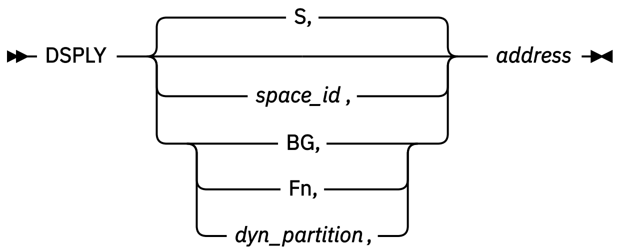
You can use DITTO/ESA for VSE to print, on a line printer, data stored on disk or tape. The program dumps the data either in character-only format or in the character and vertical hexadecimal format.

For more detailed information about DITTO/ESA for VSE, refer to the *IBM DITTO/ESA for VSE Introducing* manual.

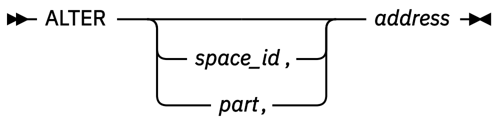
DSPLY/ALTER: Display or Alter Storage

The DSPLY command allows the console operator to display 16 bytes of virtual storage starting at the specified hexadecimal address on the device assigned to SYSLOG. Two characters (0-9, A-F) appear on SYSLOG for each byte of information; these characters represent the hexadecimal equivalent of the current information in virtual storage. You can alter this information either by the **ALTER** operator command or by using the hardware storage display feature as described under "Hardware Aids via the Operator Console" on page 227.

To request a **display** of storage, enter the command:



To **alter**, enter:



space-id
Indicates in which address space the specified address is to be displayed or altered. Valid specifications are:

0 through 9, A, B, R or S.

To display virtual storage in a **shared** area specify the space-id of any **existing** virtual address space.

BG, Fn
Indicates in which static or dynamic partition the specified address is to be displayed or altered. `part` can specify any of the static partitions BG, F1 through FB or a partition within a dynamic class, for example, P1.

address
Specifies the hexadecimal address at which the storage display or alteration is to start.

Figure 99 on page 221 shows an example of using the ALTER and DSPLY commands.

```
DSPLY 1,300
AR 015 90F21028 18215811 00185801 0014F9F9   *.2...........99*
AR 015 1I40I  READY


ALTER 1,300
AR 015 1I42D  ADDRESS WITHIN SUPERVISOR OR SVA
AR+015
15 IGNORE
AR 015 OLD DATA:  90F21028 18215811 00185801 0014F9F9   *.2...........99*
AR 015 ENTER HEX DATA (1-16 BYTES)
AR+015
15 FFF2
AR 015 1I40I


DSPLY 1,300
AR 015 FFF21028 18215811 00185801 0014F9F9   *.2...........99*
AR 015 1I40I  READY
```

Figure 99. Sample of the DSPLY and ALTER Commands

The example above shows the commands to display the contents of address X'300' in space 1 and to alter two bytes beginning with the same address to X'FFF2'. Message 1I42D is not displayed if the area you want to alter is in the user area.

For a detailed description of the ALTER or DSPLY command see z/VSE System Control Statements.

Restriction

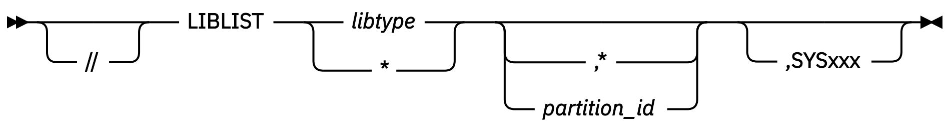
- If the specified address is within an invalid address area, the command is ignored and a corresponding information message is issued.

  If the 16 bytes to be displayed cross the boundary from a valid to an invalid address area, only the bytes in the valid address area are displayed, and a corresponding information message is issued.

  Invalid addresses are:

  - Locations beyond the end of virtual storage.
  - Unused (not allocated) partition GETVIS space.
  - A location in the page pool.
  - A location within an unallocated area of the virtual address space.
  - A location in the partition's virtual address area when a program in that partition is being executed in real mode, or vice versa.

LIBLIST: Display Library Chains



Library search chains established with the job control LIBDEF statement can be displayed either on the system console or on SYSLST with the LIBLIST job control statement.

```
libtype = Corresponds to the type operand of the
          LIBDEF statement
*       = specifies that library definitions of all
          LIBDEF statements (except DUMP) are to be
          displayed.

partition-id = static or dynamic partition whose
               library chains are to be listed.
*            = The library chains of the partition
               processing the statement are to be
               listed (default).
```

```
SYSxxx = SYSLST or SYSLOG device for output.
         Default = SYSLOG if entered from SYSLOG,
                 = SYSLST if entered from SYSRDR.
```

Figure 100 on page 222 is an example of a listing of BG partition's active library search chains resulting from a LIBLIST command.

```
// LIBLIST *,BG

TYPE: PHASE

    BG-TEMP ** NO LIBRARY INFORMATION AVAILABLE

    BG-PERM LIBNAME SUBLIB     STATUS  -PARTITIONS-  +DYNPARTS
    SEARCH  PRVLIB  TCLIB              01 34
            IJSYSRS SYSLIB             0123456789AB DYNP
    CATALOG PRVLIB1 TCLIB              0 23

TYPE: OBJ

    BG-TEMP ** NO LIBRARY INFORMATION AVAILABLE

    BG-PERM LIBNAME SUBLIB     STATUS  -PARTITIONS-
    SEARCH  PRVLIB  TCLIB              01 34
            IJSYSRS SYSLIB             0123456789AB DYNP
            PRVLIBS SLIB2      SEC SHR 0   4

TYPE: SOURCE

    BG-TEMP ** NO LIBRARY INFORMATION AVAILABLE

    BG-PERM LIBNAME SUBLIB     STATUS  -PARTITIONS-
    SEARCH  PRVLIB1 TCLIB              01 34
            IJSYSRS SYSLIB             0123456789AB DYNP
            SERVLIB S1$XE8             0

TYPE: PROC

    BG-TEMP ** NO LIBRARY INFORMATION AVAILABLE

    BG-PERM LIBNAME SUBLIB     STATUS  -PARTITIONS-
    SEARCH  PRVLIB1 TCLIB              01 34
            IJSYSRS SYSLIB             0123456789AB DYNP
```

No temporary search chain is defined in the above example.

The keyword DYNPARTS means that at least one dynamic partition has a LIBDEF to the sublibrary specified.

The device on which PRVLIBS SLIB2 resides is shared by two or more CPUs – indicated by SHR in the STATUS column. SLIB2 also is a secured (SEC) sublibrary.

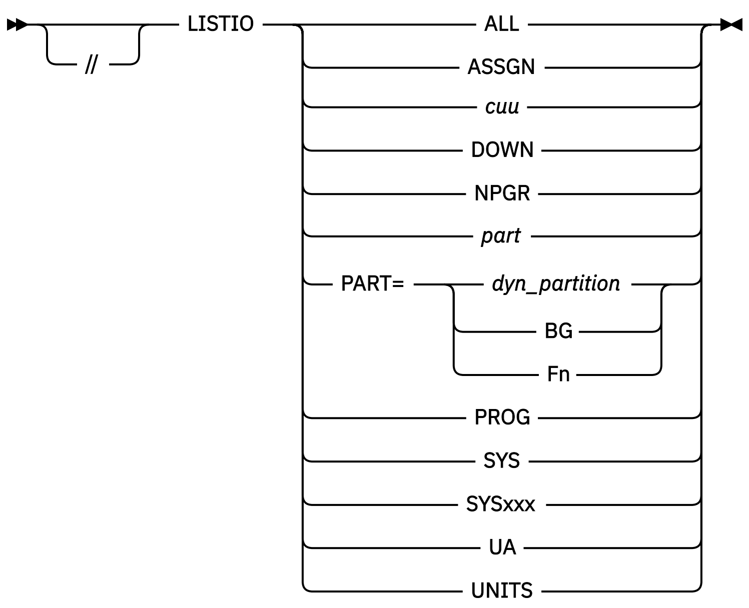
Figure 100. Example: Library Chain Listing

LIST: Print Language Translator Source Code

```
►►─ // OPTION LIST ─►◄
```

Normally, a language translator source listing is requested for all language translator runs by specifying option **LIST** in the STDOPT command as part of the IPL procedure. Should NOLIST have been specified in that command, you can override it by including // OPTION LIST in the job control statements for your language-translator run.

LISTIO: List I/O Device Assignments



You can list I/O device assignments before execution of the program begins by inserting the job control **LISTIO** statement or command in the program. The list will appear on SYSLOG if you insert the LISTIO command without //, or on SYSLST with //. The listing represents the device assignment status at the time the statement or command is being executed and not when an error occurred during a previous run.

For a detailed description of the operands of the LISTIO command, refer to z/VSE System Control Statements.

LISTLOG: Display Console Communication

►►— // EXEC LISTLOG —►◄

You can use the LISTLOG utility program to request a listing of all information collected for a specific job in the hard copy file. The program writes this list to the device assigned to SYSLST.

You invoke the program by inserting the statement // EXEC LISTLOG immediately following the /& statement for the job. z/VSE invokes the program automatically whenever a job is canceled.

The printout provided by the LISTLOG utility program lists:

- Job control statements submitted for the job,
- All messages displayed on the console for this job,
- Any attention routine messages and commands that occurred while the job was being executed,
- Operator responses.

LOG: Print Job Control Statements

The LOG command or statement causes the system to log all job control commands and statements.

►►— // LOG —►◄

►►— // OPTION LOG —►◄

The // LOG statement has the same effect as the // OPTION LOG statement.

►►— // NOLOG —►◄

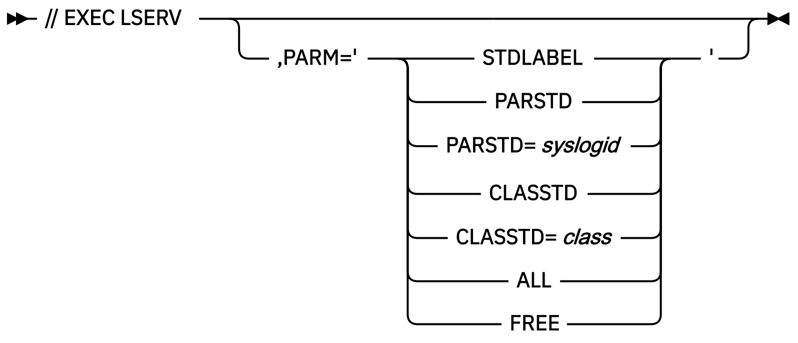
The two statements are interchangeable, and each can be reset by either the // NOLOG or the // OPTION NOLOG statement.

For a detailed description of the LOG statement, refer to z/VSE System Control Statements.

LSERV: Display Label Information Area

LSERV, a system utility program, produces a printout of the label information area on the device assigned to SYSLST.

The job control statement for LSERV is as follows:



The following is a description of the parameters used in the job control statement:

STDLABEL
prints the system standard labels only.

PARSTD
prints the partition standard labels of all active partitions.

PARSTD=syslogid
prints the partition standard labels of the specified partition only.

CLASSTD
prints all class standard labels only.

CLASSTD=class
prints the class standard labels of the specified class only.

ALL
prints all labels, including free-usage labels. In addition, label information for data secured files (see also the DSF operand in the DLBL statement) is displayed.

FREE
prints all free-usage labels only.

If no parameter is specified, all labels (but not DSF labels and free-usage labels) are printed. User labels from static or dynamic partitions are only included when no parameter or ALL is specified. User labels change from job to job and, therefore, no special support is needed in the LSERV program.

A sample, partial output of an LSERV run for the above control statements, is shown in the example in Figure 101 on page 225. The output shows label-information records contained in different label groups, such as system standard labels, class standard labels, or partition labels. Each active partition (static or dynamic) can establish up to three label groups: free-usage, temporary, and permanent partition labels.

The output shows the relationship between job control DLBL and EXTENT statements. Fur further information, refer to the z/VSE Guide to System Functions.

For VSAM files only

There is an additional label information record following the VSAM label record if, in the DLBL statement, at least one of the operands DISP, RECORDS, or RECSIZE is specified.

Note: A warning message is issued on SYSLST if you request LSERV while another partition is updating the label area. The free-usage label groups which are used internally by VSE/ICCF and vendor software, are only shown with PARM='ALL' or PARM='FREE'.

```
                               LABEL INFORMATION DISPLAY   PAGE nnn
EXAMPLE
    FILE IDENTIFIER                          EXAMPLE
    FILE SERIAL NUMBER                       OMITTED
    VOLUME SEQUENCE NUMBER                   01
    CREATION DATE                            OMITTED
    RETENTION PERIOD (DAYS)                  0999
    FILE TYPE                                SEQUENTIAL

  EXTENT INFORMATION
    EXTENT SEQUENCE NUMBER                   000
    EXTENT TYPE                              1 (PRIME DATA)
    RELATIVE START ADDRESS                   002
    NUMBER OF TRACKS/BLOCKS                  045107
    SYMBOLIC UNIT                            SYSRES LOGICAL UNIT FORMAT
                                             TYP=00,NUM=06
    VOLUME SERIAL NUMBER                     OMITTED

  ADDITIONAL INFORMATION
    DISPOSITION                              (OLD,KEEP)
    RECORDS                                  (0000000500,0000000100)
    RECORD SIZE                              0000000080

SALARY
    FILE IDENTIFIER                          SALARY.1999.FILE
    FILE SERIAL NUMBER                       DASD02
    VOLUME SEQUENCE NUMBER                   01
    CREATION DATE                            OMITTED
    EXPIRATION DATE                          11/365
    FILE TYPE                                SEQUENTIAL

  EXTENT INFORMATION
    EXTENT SEQUENCE NUMBER                   000
    EXTENT TYPE                              1 (PRIME DATA)
    RELATIVE START ADDRESS IN TRACKS/BLOCKS 010000
    NUMBER OF TRACKS/BLOCKS                  001000
    SYMBOLIC UNIT                            SYS019  LOGICAL UNIT FORMAT
                                             TYP=01,NUM=13
    VOLUME SERIAL NUMBER                     DASD02
```

Figure 101. Sample: LSERV Output

LVTOC: Display Volume Table of Contents

```
// ASSGN SYS004,cuu
// ASSGN SYS005,cuu
// EXEC LVTOC
```

A volume table of contents (VTOC) is an index of all files, and the remaining space, on a disk volume. A VTOC display can be requested by executing the LVTOC program with SYS004 assigned to the applicable disk drive and SYS005 to a printer. LVTOC lists the file labels contained in a VTOC in alphabetic sequence by file name. It also provides a listing of free space on the volume, with the start and end addresses and sizes of the unused space. The control statements needed to invoke that program may be submitted via SYSRDR or via the console as shown in Table 17 on page 226.

A display of a VTOC can be requested also in response to messages. Such a response is CANCELV or DSPLYV. Use CANCELV if you intend to cancel the job, or DSPLYV if the condition allows program execution to be continued after the VTOC display.

Table 17. Control Statements to Invoke LVTOC

Submission via SYSRDR	Submission via the console

// JOB anyname
// ASSGN SYS004,cuu (A)
// ASSGN SYS005,cuu (B)
// EXEC LVTOC
/&
	1. Press the Request key
	2. Enter:
	PAUSE p.-id,EOJ (C)
	3. Wait for end-of-job in the specified partition.
	4. Enter:
	// ASSGN SYS004,cuu (A)
	// ASSGN SYS005,cuu (B)
	// EXEC LVTOC

(A)
The unit address of the disk for which the VTOC is desired.

(B)
The unit address of the device on which the VTOC is to be listed (normally a line printer).

(C)
The identifier of the partition (F1 ...) in which LVTOC is to run.

STOP/PAUSE: Suspend Program Execution

```
STOP/PAUSE
```

Suspending program execution between job steps can be of much help during hands-on diagnosis.

You suspend program execution with the **STOP** command either via the console or, if you use a card reader, via SYSRDR. Another possibility is to submit the job control **PAUSE** statement or command. Both methods result in program execution to be suspended when job control executes the statement or command. The PAUSE command is used to interrupt the execution of the job. The operator may enter additional job control statements via SYSLOG at this time. The STOP command removes the partition from the system's task selection mechanism and no read is issued to the SYSLOG or SYSRDR device for that partition.

To resume program execution after a STOP command, issue an attention routine **START** command for the partition. To resume program execution after a PAUSE statement or command, simply press END/ENTER. Note that the STOP and START commands can be given in a static partition only. For dynamic partitions, use the CANCEL or VSE/POWER PFLUSH command.

Appendix C. Hardware Service Aids

Controlling the Recovery Management Support

The recording activity of RMS can be controlled via the operator command ROD.

Use the **ROD** command to:

- Add error statistics to the system recorder file.
- Have RMS write MDR records into the SYSREC file for those devices that are equipped with an internal error log.
- Have RMS build an end-of-day (EOD) record and write this record on SYSREC.
- Write the hardcopy buffer into the hardcopy file.

The ROD command is discussed in more detail in the following section.

The ROD Command

```
►► ROD ►◄
```

The ROD command has no operands. Issuing the ROD command causes the hardcopy buffer written to the hardcopy file and RMS to record, on SYSREC:

- Error statistics that were compiled for I/O devices (except telecommunication devices).
- An end-of-day record if RMS received an appropriate response to a prompting message via the console.

Retrieval and Analysis of RMS Information

The EREP Program

For the retrieval of information recorded by RMS on SYSREC, use the IBM EREP program. How to use this program is described in the separate publications, *EREP User's Guide* and *EREP Reference*.

For a number of VSE messages, the recommended response in z/VSE Messages and Codes Volume 2 includes instructions to run EREP.

Hardware Aids via the Operator Console

Current IBM processors provide a variety of hardware aids for hands-on diagnosis. The procedures for using these aids are, for the most part, processor-model dependent, and are described in detail in the operating procedures manuals for these processors. Therefore, this section discusses only aspects such as usefulness of the aids, when to use them, and requirements or precautions for their use.

⚠ **CAUTION:** If you use one of the following hardware serviceability and debugging aids, you interfere with normal processing under VSE. Consider using these aids only (with your local management approval) in situations such as total system failure or a hard wait condition with no VSE-supported recovery possible.

The most important hardware aids available via the operator's console for system service and program diagnosis are:

- Alter/display feature.
- Instruction stepping feature.
- Stop on address compare feature.

Hardware Alter/Display

With the alter/display feature you can display the contents of storage areas and registers as indicated below. You can also alter any of these storage areas.

Note: The alter/display feature can be used only from the operator's console of your processor; the feature is not available, for example, from a channel-attached IBM 3277 that you use as an operator's console.

Following is a list of storage areas (and registers) that you can display and alter by using this hardware aid:

- Any selected area of real or virtual storage.
- Contents of the general purpose registers.
- Contents of the floating point registers.
- Contents of the control registers.
- Current PSW.
- Storage protection key.

For detailed information on how to use this feature and on the areas that you can display or alter from your processor's console, refer to IBM's operating procedures publication for your central processor.

Instruction Stepping Feature

With the instruction stepping feature you can check and record the address of each instruction that is executed during program operation. By combined application of this feature and the alter/display feature, you can trace, for example, a short program loop. This approach of tracing executable code of a program is indicated when only short sections of code are to be traced or if, for any reason, the SDAID tracing facility cannot be used.

Refer to the operations publication for your processor for details of this feature.

Stop-on-Address-Compare Feature

This feature is provided primarily for IBM service personnel. It enables one, for example, to stop all system activity at a selected instruction address within a program. In combination with the alter/display feature (or commands ALTER, DSPLY, or DUMP), the stop-on-address-compare feature can be used to display or alter the contents of storage at this selected address. The feature can be used, for example, if there is a need for a dump of a specific area of virtual storage at a specific point of program execution.

Another use of this feature is the generation of a sync signal at a certain instruction address (this is primarily a hardware service aid).

For more information refer to the operating procedures publication for your processor.

Notices

Programming Interface Information

This publication documents intended Programming Interfaces that allow the customer to write programs to obtain services of z/VSE.

Trademarks

Terms and Conditions for Product Documentation

Permissions for the use of these publications are granted subject to the following terms and conditions.

Trademarks

Accessibility

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

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

Using Assistive Technologies

Assistive technology products, such as screen readers, function with the user interfaces found in z/VSE. Consult the assistive technology documentation for specific information when using such products to access z/VSE interfaces.

Documentation Format

The publications for this product are in Adobe Portable Document Format (PDF) and should be compliant with accessibility standards. If you experience difficulties when you use the PDF files and want to request a web-based format for a publication, you can either write an email to s390id@de.ibm.com, or use the Reader Comment Form in the back of this publication or direct your mail to the following address:

```
IBM Deutschland Research & Development GmbH
Department 3282
Schoenaicher Strasse 220
D-71032 Boeblingen
Federal Republic of Germany
```

In the request, be sure to include the publication number and title.

When you send information to IBM, you grant IBM a nonexclusive right to use or distribute the information in any way it believes appropriate without incurring any obligation to you.

Glossary

This glossary includes terms and definitions for IBM z/VSE.

The following cross-references are used in this glossary:

1. See refers the reader from a term to a preferred synonym, or from an acronym or abbreviation to the defined full form.
2. See also refers the reader to a related or contrasting term.

A

Access Control Logging and Reporting

An IBM licensed program to log all attempts of access to protected data and to print selected formatted reports on such attempts.

access control table (DTSECTAB)

A table that is used by the system to verify a user's right to access a certain resource.

access list

A table in which each entry specifies an address space or data space that a program can reference.

access method

A program, that is, a set of commands (macros) to define files or addresses and to move data to and from them; for example VSE/VSAM or VTAM.

account file

A disk file that is maintained by VSE/POWER containing accounting information that is generated by VSE/POWER and the programs running under VSE/POWER.

addressing mode (AMODE)

A program attribute that refers to the address length that a program is prepared to handle on entry. Addresses can be either 24 bits, 31 bits, or 64 bits in length. In 24 bit addressing mode, the processor treats all virtual addresses as 24-bit values; in 31 bit addressing mode, the processor treats all virtual addresses as 31-bit values and in 64-bit addressing mode, the processor treats all virtual addresses as 64-bit values. Programs with an addressing mode of ANY can receive control in either 24 bit or 31 bit addressing mode. 64 bit addressing mode cannot be used as program attribute.

administration console

In z/VSE, one or more consoles that receive all system messages, except for those that are directed to one particular console. Contrast this with the user console, which receives only those messages that are directed to it, for example messages that are issued from a job that was submitted with the request to echo its messages to that console. The operator of an administration console can reply to all outstanding messages and enter all system commands.

alternate block

On an FBA disk, a block that is designated to contain data in place of a defective block.

alternate index

In systems with VSE/VSAM, the index entries of a given base cluster that is organized by an alternate key, that is, a key other than the prime key of the base cluster. For example, a personnel file preliminary ordered by names can be indexed also by department number.

alternate library

An interactively accessible library that can be accessed from a terminal when the user of that terminal issues a connect or switch library request.

alternate track

A library, which becomes accessible from a terminal when the user of that terminal issues a connect or switch (library) request.

AMODE

Addressing mode.

APA

All points addressable.

APAR

Authorized Program Analysis Report.

appendage routine

A piece of code that is physically located in a program or subsystem, but logically and extension of a supervisor routine.

application profile

A control block in which the system stores the characteristics of one or more application programs.

application program

A program that is written for or by a user that applies directly to the user's work, such as a program that does inventory control or payroll. See also batch program and online application program.

AR/GPR

Access register and general-purpose register pair.

ASC mode

Address space control mode.

ASI (automated system initialization) procedure

A set of control statements, which specifies values for an automatic system initialization.

attention routine (AR)

A routine of the system that receives control when the operator presses the Attention key. The routine sets up the console for the input of a command, reads the command, and initiates the system service that is requested by the command.

automated system initialization (ASI)

A function that allows control information for system startup to be cataloged for automatic retrieval during system startup.

autostart

A facility that starts VSE/POWER with little or no operator involvement.

auxiliary storage

Addressable storage that is not part of the processor, for example storage on a disk unit. Synonymous with external storage.

B

B-transient

A phase with a name beginning with $$B and running in the Logical Transient Area (LTA). Such a phase is activated by special supervisor calls.

bar

2 GigyByte (GB) line

basic telecommunications access method (BTAM)

An access method that permits read and write communication with remote devices. BTAM is not supported on z/VSE.

BIG-DASD

A subtype of Large DASD that has a capacity of more than 64 K tracks and uses up to 10017 cylinders of the disk.

block

Usually, a block consists of several records of a file that are transmitted as a unit. But if records are very large, a block can also be part of a record only. On an FBA disk, a block is a string of 512 bytes of data. See also a control block.

block group

In VSE/POWER, the basic organizational unit for fixed-block architecture (FBA) devices. Each block group consists of a number of 'units of transfer' or blocks.

C

CA splitting

Is the host part of the VSE JavaBeans, and is started using the job STARTVCS, which is placed in the reader queue during installation of z/VSE. Runs by default in dynamic class R. In VSE/VSAM, to double a control area dynamically and distribute its CIs evenly when the specified minimum of free space get used up by more data.

carriage control character

The fist character of an output record (line) that is to be printed; it determines how many lines should be skipped before the next line is printed.

catalog

A directory of files and libraries, with reference to their locations. A catalog may contain other information such as the types of devices in which the files are stored, passwords, blocking factors. To store a library member such as a phase, module, or book in a sublibrary. See also VSE/VSAM catalog.

cell pool

An area of virtual storage that is obtained by an application program and managed by the callable cell pool services. A cell pool is located in an address space or a data space and contains an anchor, at least one extent, and any number of cells of the same size.

central location

The place at which a computer system's control device, normally the systems console in the computer room, is installed.

chained sublibraries

A facility that allows sublibraries to be chained by specifying the sequence in which they must be searched for a certain library member.

chaining

A logical connection of sublibraries to be searched by the system for members of the same type (phases or object modules, for example).

channel command word (CCW)

A doubleword at the location in main storage that is specified by the channel address word. One or more CCWs make up the channel program that directs data channel operations.

channel program

One or more channel command words that control a sequence of data channel operations. Execution of this sequence is initiated by a start subchannel instruction.

channel scheduler

The part of the supervisor that controls all input/output operations.

channel subsystem

A feature of z/Architecture that provides extensive additional channel (I/O) capabilities to IBM Z.

channel to channel attachment (CTCA)

A function that allows data to be exchanged

1. Under the control of VSE/POWER between two virtual VSE machines running under VM or
2. Under the control of VTAM between two processors.

character-coded request

A request that is encoded and transmitted as a character string. Contrast with *field-formatted request*.

checkpoint

1. A point at which information about the status of a job and the system can be recorded so that the job step can be restarted later.
2. To record such information.

CICS (Customer Information Control System)

An IBM program that controls online communication between terminal users and a database. Transactions that are entered at remote terminals are processed concurrently by user-written application programs. The program includes facilities for building, using, and servicing databases.

CICS ECI

The CICS External Call Interface (ECI) is one possible requester type of the *CICS business logic interface* that is provided by the CICS Transaction Server for z/VSE. It is part of the CICS client and allows workstation programs to CICS function on the z/VSE host.

CICS EXCI

The EXternal CICS Interface (EXCI) is one possible requester type of the *CICS business logic interface* that is provided by the CICS Transaction Server for z/VSE. It allows any BSE batch application to call CICS functions.

CICS system definition data set (CSD)

A VSAM KSDS cluster that contains a resource definition record for every record defined to CICS using resource definition online (RDO).

CICS Transaction Server for z/VSE

A z/VSE base program that controls online communication between terminal users and a database. This is the successor system to CICS/VSE.

CICS TS

CICS Transaction Server

CICS/VSE

Customer Information Control System/VSE. No longer shipped on the Extended Base Tape and no longer supported, cannot run on z/VSE 5.1 or later.

class

In VSE/POWER, a group of jobs that either come from the same input device or go to the same output device.

CMS

Conversational monitor system running on z/VM.

common library

A library that can be interactively accessed by any user of the (sub)system that owns the library.

communication adapter

A circuit card with associated software that enables a processor, controller, or other device to be connected to a network.

communication region

An area of the supervisor that is set aside for transfer of information within and between programs.

component

1. Hardware or software that is part of a computer system.
2. A functional part of a product, which is identified by a component identifier.
3. In z/VSE, a component program such as VSE/POWER or VTAM.
4. In VSE/VSAM, a named, cataloged group of stored records, such as the data component or index component of a key-sequenced file or alternate index.

component identifier

A 12-byte alphanumeric string, uniquely defining a component to MSHP.

conditional job control

The capability of the job control program to process or to skip one or more statements that are based on a condition that is tested by the program.

connect

To authorize library access on the lowest level. A modifier such as "read" or "write" is required for the specified use of a sublibrary.

connection pooling

Introduced with an z/VSE 5.1 update to manage (reuse) connections of the z/VSE database connector in CICS TS.

connector

In the context of z/VSE, a connector provides the middleware to connect two platforms: Web Client and z/VSE host, middle-tier and z/VSE host, or Web Client and middle-tier.

connector (e-business connector)

A piece of software that is provided to connect to heterogeneous environments. Most connectors communicate to non-z/VSE Java-capable platforms.

container

Is part of the JVM of application servers such as the IBM WebSphere Application Server, and facilitates the implementation of servlets, EJBs, and JSPs, by providing resource and transaction management resources. For example, an EJB developer must not code against the JVM of the application server, but instead against the interface that is provided by the container. The main role of a container is to act as an intermediary between EJBs and clients, Is the host part of the VSE JavaBeans, and is started using the job STARTVCS, which is placed in the reader queue during the installation of z/VSE. Runs by default in dynamic class R. and also to manage multiple EJB instances. After EJBs have been written, they must be stored in a container residing on an application server. The container then manages all threading and client-interactions with the EJBs, and co-ordinate connection- and instance pooling.

control interval (CI)

A fixed-length area of disk storage where VSE/VSAM stores records and distributes free space. It is the unit of information that VSE/VSAM transfers to or from disk storage. For FBA it must be an integral multiple to be defined at cluster definition, of the block size.

control program

A program to schedule and supervise the running of programs in a system.

conversational monitor system (CMS)

A virtual machine operating system that provides general interactive time sharing, problem solving, and program development capabilities and operates under the control of z/VM.

count-key-data (CKD) device

A disk device that store data in the record format: count field, key field, data field. The count field contains, among others, the address of the record in the format: cylinder, head (track), record number, and the length of the data field. The key field, if present, contains the record's key or search argument. CKD disk space is allocated by tracks and cylinders. Contrast with *FBA disk device. See also extended count-key-data device.*

cross-partition communication control

A facility that enables VSE subsystems and user programs to communicate with each other; for example, with VSE/POWER.

cryptographic token

Usually referred to simply as a *token*, this is a device, which provides an interface for performing cryptographic functions like generating digital signatures or encrypting data.

cryptography

1. A method for protecting information by transforming it (encrypting it) into an unreadable format, called ciphertext. Only users who possess a secret key can decipher (or decrypt) the message into plaintext.
2. The transformation of data to conceal its information content and to prevent its unauthorized use or undetected modification .

D

data block group

The smallest unit of space that can be allocated to a VSE/POWER job on the data file. This allocation is independent of any device characteristics.

data conversion descriptor file (DCDF)

With a DCDF, you can convert individual fields within a record during data transfer between a PC and its host. The DCDF defines the record fields of a particular file for both, the PC and the host environment.

data import

The process of reformatting data that was used under one operating system such that it can subsequently be used under a different operating system.

Data Interfile Transfer, Testing, and Operations (DITTO) utility

An IBM program that provides file-to-file services for card I/O, tape, and disk devices. The latest version is called DITTO/ESA for VSE.

Data Language/I (DL/I)

A database access language that is used with CICS.

data link

In SNA, the combination of the link connection and the link stations joining network noes, for example, a z/Architecture channel and its associated protocols. A link is both logical and physical.

data security

The protection of data against unauthorized disclosure, transfer, modification, or destruction, whether accidental or intentional .

data set header record

In VSE/POWER abbreviated as DSHR, alias NDH or DSH. An NJE control record either preceding output data or, in the middle of input data, indicating a change in the data format.

data space

A range of up to 2 gigabytes of contiguous virtual storage addresses that a program can directly manipulate through z/Architecture instructions. Unlike an address space, a data space can hold only user data; it does not contain shared areas, or programs. Instructions do not execute in a data space. Contrast with address space.

data terminal equipment (DTE)

In SNA, the part of a data station that serves a data source, data sink, or both.

database connector

Is a function introduced with z/VSE 5.1.1, which consists of a client and server part. The client provides an API (CBCLI) to be used by applications on z/VSE, the server on any Java capable platform connects a JDBC driver that is provided by the database. Both client and server communicate via TCP/IP.

Database 2 (Db2)

An IBM rational database management system.

Db2-based connector

Is a feature introduced with VSE/ESA 2.5, which includes a customized Db2 version, together with VSAM and DL/I functionality, to provide access to Db2, VSAM, and DL/I data, using Db2 Stored Procedures.

Db2 Runtime only Client edition

The Client Edition for z/VSE comes with some enhanced features and improved performance to integrate z/VSE and Linux on z Systems.

Db2 Stored Procedure

In the context of z/VSE, a Db2 Stored Procedure is a Language Environment (LE) program that accesses Db2 data. However, from VSE/ESA 2.5 onwards you can also access VSAM and DL/I data using a Db2 Stored Procedure. In this way, it is possible to exchange data between VSAM and Db2.

DBLK

Data block.

DCDF

Data conversion descriptor file.

deblocking

The process of making each record of a block available for processing.

dedicated (disk) device

A device that cannot be shared among users.

device address

1. The identification of an input/output device by its device number.
2. In data communication, the identification of any device to which data can be sent or from which data can be received.

device driving system (DDS)

A software system external to VSE/POWER, such as a CICS spooler or PSF, that writes spooled output to a destination device.

Device Support Facilities (DSF)

An IBM supplied system control program for performing operations on disk volumes so that they can be accessed by IBM and user programs. Examples of these operations are initializing a disk volume and assigning an alternative track.

device type code

The four- or five-digit code that is used for defining an I/O device to a computer system. See also ICKDSF

dialog

In an interactive system, a series of related inquiries and responses similar to a conversation between two people. For z/VSE, a set of panels that can be used to complete a specific task; for example, defining a file.

dialog manager

The program component of z/VSE that provides for ease of communication between user and system.

digital signature

In computer security, encrypted data, which is appended to or part of a message, that enables a recipient to prove the identity of the sender.

Digital Signature Algorithm (DSA)

The Digital Signature Algorithm is the US government-defined standard for digital signatures. The DSA digital signature is a pair of large numbers, computed using a set of rules (that is, the DSA) and a set of parameters such that the identity of the signatory and integrity of the data can be verified. The DSA provides the capability to generate and verify signatures.

directory

In z/VSE the index for the program libraries.

direct access

Accessing data on a storage device using their address and not their sequence. This is the typical access on disk devices as opposed to magnetic tapes. Contrast with *sequential access*.

disk operating system residence volume (DOSRES)

The disk volume on which the system sublibrary IJSYSRS.SYSLIB is located including the programs and procedures that are required for system startup.

disk sharing

An option that lets independent computer systems uses common data on shared disk devices.

disposition

A means of indicating to VSE/POWER how a job input or output entry is to be handled: according to its local disposition in the RDR/LST/PUN queue or its transmission disposition when residing in the XMT queue. A job might, for example, be deleted or kept after processing.

distribution tape

A magnetic tape that contains, for example, a preconfigured operating system like z/VSE. This tape is shipped to the customer for program installation.

DITTO/ESA for VSE

Data Interfile Transfer, Testing, and Operations utility. An IBM program that provides file-to-file services for disk, tape, and card devices.

DSF

Device Support Facilities.

DSH (R)

Data set header record.

dummy device

A device address with no real I/O device behind it. Input and output for that device address are spooled on disk.

duplex

Pertaining to communication in which data can be sent and received at the same time.

DU-AL (dispatchable unit - access list)

The access list that is associated with a z/VSE main task or subtask. A program uses the DU-AL associated with its task and the PASN-AL associated with its partition. See also "PASN-AL (primary address space number - access list)" on page 256.

dynamic class table

Defines the characteristics of dynamic partitions.

dynamic partition

A partition that is created and activated on an 'as needed' basis that does not use fixed static allocations. After processing, the occupied space is released. Dynamic partitions are grouped by class, and jobs are scheduled by class. Contrast with *static partition*.

dynamic space reclamation

A librarian function that provides for space that is freed by the deletion of a library member to become reusable automatically.

E

ECI

See “CICS ECI” on page 239.

emulation

The use of programming techniques and special machine features that permit a computer system to execute programs that are written for another system or for the use of I/O devices different from those that are available.

emulation program (EP)

An IBM control program that allows a channel-attached 3705 or 3725 communication controller to emulate the functions of an IBM 2701 Data Adapter Unit, or an IBM 2703 Transmission Control.

end user

1. A person who makes use of an application program.
2. In SNA, the ultimate source or destination of user data flowing through an SNA network. Might be an application program or a terminal operator.

Enterprise Java Bean

An EJB is a distributed bean. "Distributed" means, that one part of an EJB runs inside the JVM of a web application server, while the other part runs inside the JVM of a web browser. An EJB either represents one data row in a database (entity bean), or a connection to a remote database (session bean). Normally, both types of an EJB work together. This allows to represent and access data in a standardized way in heterogeneous environments with relational and non-relational data. See also *JavaBean*.

entry-sequenced file

A VSE/VSAM file whose records are loaded without respect to their contents and whose relative byte addresses cannot change. Records are retrieved and stored by addressed access, and new records are added to the end of the file.

Environmental Record Editing and Printing (EREP) program

A z/VSE base program that makes the data that is contained in the system record file available for further analysis.

EPI

See *CICS EPI.*

ESCON Channel (Enterprise Systems Connection Channel)

A serial channel, using fiber optic cabling, that provides a high-speed connection between host and control units for I/O devices. It complies with the ESA/390 and IBM Z I/O Interface until z114. The zEC12 processors do not support ESCON channels.

exit routine

1. Either of two types of routines: installation exit routines or user exit routines. Synonymous with exit program.
2. See *user exit routine*.

extended addressability

The ability of a program to use 31 bit or 64 bit virtual storage in its address space or outside the address space.

extended recovery facility (XRF)

In z/VSE, a feature of CICS that provides for enhanced availability of CICS by offering one CICS system as a backup of another.

External Security Manager (ESM)

A priced vendor product that can provide extended functionality and flexibility that is compared to that of the Basic Security Manager (BSM), which is part of z/VSE.

F

FASTCOPY

See "VSE/Fast Copy" on page 267.

fast copy data set program (VSE/Fast Copy)

See "VSE/Fast Copy" on page 267.

fast service upgrade (FSU)

A service function of z/VSE for the installation of a refresh release without regenerating control information such as library control tables.

FAT-DASD

A subtype of Large DASD, it supports a device with more than 4369 cylinders (64 K tracks) up to 64 K cylinders.

FCOPY

See *VSE/Fast Copy.*

fence

A separation of one or more components or elements from the remainder of a processor complex. The separation is by logical boundaries. It allows simultaneous user operations and maintenance procedures.

fetch

1. To locate and load a quantity of data from storage.

2. To bring a program phase into virtual storage from a sublibrary and pass control to this phase.
3. The name of the macro instruction (FETCH) used to accomplish 2. See also *loader.*

Fibre Channel Protocol (FCP)

A combination of hardware and software conforming to the Fibre Channel standards and allowing system and peripheral connections via FICON and FICON Express feature cards on IBM zSeries processors. In z/VSE, zSeries FCP is employed to access industry-standard SCSI disk devices.

fragmentation (of storage)

Inability to allocate unused sections (fragments) of storage in the real or virtual address range of virtual storage.

FSU

Fast service upgrade.

FULIST (FUnction LIST)

A type of selection panel that displays a set of files and/or functions for the choice of the user.

G

generation

See *macro generation.*

generation feature

An IBM licensed program order option that is used to tailer the object code of a program to user requirements.

GETVIS space

Storage space within partition or the shared virtual area, available for dynamic allocation to programs.

guest system

A data processing system that runs under control of another (host) system. On the mainframe z/VSE can run as a guest of z/VM.

H

hard wait

The condition of a processor when all operations are suspended. System recovery from a hard wait is impossible without performing a new system startup.

hash function

A hash function is a transformation that takes a variable-size input and returns a fixed-size string, which is called the hash value. In cryptography, the hash functions should have some additional properties:

- The hash function should be easy to compute.
- The hash function is one way; that is, it is impossible to calculate the 'inverse' function.

- The hash function is collision-free; that is, it is impossible that different input leads to the same hash value.

hash value

The fixed-sized string resulting after applying a *hash function* to a text.

High-Level Assembler for VSE

A programming language providing enhanced assembler programming support. It is a base program of z/VSE.

home interface

Provides the methods to instantiate a new EJB object, introspect an EJB, and remove an EJB instantiation., as for the remote interface is needed because the deployment tool generates the implementation class. Every Session bean's home interface must supply at least one *create()* method.

host mode

In this operating mode, a PC can access a VSE host. For programmable workstation (PWS) functions, the Move Utilities of VSE can be used.

host system

The controlling or highest level system in a data communication configuration.

host transfer file (HTF)

Used by the Workstation File Transfer Support of z/VSE as an intermediate storage area for files that are sent to and from IBM personal computers.

HTTP Session

In the context of z/VSE, identifies the web-browser client that calls a servlet (in other words, identifies the connection between the client and the middle-tier platform).

I

ICCF

See *VSE/ICCF.*

ICKDSF (Device Support Facilities)

A z/VSE base program that supports the installation, use, and maintenance of IBM disk devices.

include function

Retrieves a library member for inclusion in program input.

index

1. A table that is used to locate records in an indexed sequential data set or on indexed file.
2. In, an ordered collection of pairs, each consisting of a key and a pointer, used by to sequence and locate the records of a key-sequenced data set or file; it is organized in levels of index records. See also *alternate index.*

input/output control system (IOCS)

A group of IBM supplied routines that handle the transfer of data between main storage and auxiliary storage devices.

integrated communication adapter (ICA)

The part of a processor where multiple lines can be connected.

integrated console

In z/VSE, the service processor console available on IBM Z that operates as the z/VSE system console. The integrated console is typically used during IPL and for recovery purposes when no other console is available.

Interactive Computing and Control Facility (ICCF)

An IBM licensed program that serves as interface, on a time-slice basis, to authorized users of terminals that are linked to the system's processor.

interactive partition

An area of virtual storage for the purpose of processing a job that was submitted interactively via VSE/ICCF.

Interactive User Communication Vehicle (IUCV)

Programming support available in a VSE supervisor for operation under z/VM. The support allows users to communicate with other users or with CP in the same way they would with a non-preferred guest.

intermediate storage

Any storage device that is used to hold data temporarily before it is processed.

IOCS

Input/output control system.

IPL

Initial program load.

irrecoverable error

An error for which recovery is impossible without the use of recovery techniques external to the computer program or run.

IUCV

Interactive User Communication Vehicle.

J

JAR

Is a platform-independent file format that aggregates many files into one. Multiple applets and their requisite components (.class files, images, and sounds) can be bundled in a JAR file, and then downloaded to a web browser using a single HTTP transaction (much improving the download speed). The JAR format also supports compression, which reduces the files size (and further improves the download speed). The compression algorithm that is used is fully compatible with the ZIP algorithm. The owner of an applet can also digitally sign individual entries in a JAR file to authenticate their origin.

Java application

A Java program that runs inside the JVM of your web browser. The program's code resides on a local hard disk or on the LAN. Java applications might be large programs using graphical interfaces. Java applications have unlimited access to all your local resources.

Java bytecode

Bytecode is created when a file containing Java source language statements is compiled. The compiled Java code or "bytecode" is similar to any program module or file that is ready to be executed (run on a computer so that instructions are performed one at a time). However, the instructions in the bytecode are really instructions to the *Java Virtual Machine*. Instead of being interpreted one instruction at a time, bytecode is instead recompiled for each operating-system platform using a just-in-time (JIT) compiler. Usually, this enables the Java program to run faster. Bytecode is contained in binary files that have the suffix**.CLASS**

Java servlet

See *servlet.*

JHR

Job header record.

job accounting interface

A function that accumulates accounting information for each job step, to be used for charging the users of the system, for planning new applications, and for supervising system operation more efficiently.

job accounting table

An area in the supervisor where accounting information is accumulated for the user.

job catalog

A catalog made available for a job by means of the file name IJSYSUC in the respective DLBL statement.

job entry control language (JECL)

A control language that allows the programmer to specify how VSE/POWER should handle a job.

job step

In 1 of a group of related programs complete with the JCL statements necessary for a particular run. Every job step is identified in the job stream by an EXEC statement under one JOB statement for the whole job.

job trailer record (JTR)

As VSE/POWER parameter JTR, alias NJT. An NJE control record terminating a job entry in the input or output queue and providing accounting information.

K

key

In VSE/VSAM, one or several characters that are taken from a certain field (key field) in data records for identification and sequence of index entries or of the records themselves.

key sequence

The collating sequence either of records themselves or of their keys in the index or both. The key sequence is alphanumeric.

key-sequenced file

A VSE/VSAM file whose records are loaded in key sequence and controlled by an index. Records are retrieved and stored by keyed access or by addressed access, and new records are inserted in the file in key sequence.

KSDS

Key-sequenced data sets. See *key-sequenced file.*

L

label

1. An identification record for a tape, disk, or diskette volume or for a file on such a volume.
2. In assembly language programming, a named instruction that is generally used for branching.

label information area

An area on a disk to store label information that is read from job control statements or commands. Synonymous with *label area*.

Language Environment for z/VSE

An IBM software product that is the implementation of Language Environment on the VSE platform.

language translator

A general term for any assembler, compiler, or other routine that accepts statements in one language and produces equivalent statements in another language.

Large DASD

A DASD device that

1. Has a capacity exceeding 64 K tracks and
2. Does not have VSAM space created prior to VSE/ESA 2.6 that is owned by a catalog.

LE/VSE

Short form of Language Environment for z/VSE.

librarian

The set of programs that maintains, services, and organizes the system and private libraries.

library block

A block of data that is stored in a sublibrary.

library directory

The index that enables the system to locate a certain sublibrary of the accessed library.

library member

The smallest unit of a data that can be stored in and retrieved from a sublibrary.

line commands

In VSE/ICCF, special commands to change the declaration of individual lines on your screen. You can copy, move, or delete a line declaration, for example.

linkage editor

A program that is used to create a phase (executable code) from one or more independently translated object modules, from one or more existing phases, or from both. In creating the phase, the linkage editor resolves cross-references among the modules and phases available as input. The program can catalog the newly built phases.

linkage stack

An area of protected storage that the system gives to a program to save status information for a branch and stack or a stacking program call.

link station

In SNA, the combination of hardware and software that allows a node to attach to and provide control for a link.

loader

A routine, commonly a computer program, that reads data or a program into processor storage. See also *relocating loader*.

local shared resources (LSR)

A VSE/VSAM option that is activated by three extra macros to share control blocks among files.

lock file

In a shared disk environment under VSE, a system file on disk that is used by the sharing systems to control their access to shared data.

logical partition

In LPAR mode, a subset of the server unit hardware that is defined to support the operation of a system control program.

logical record

A user record, normally pertaining to a single subject and processed by data management as a unit. Contrast with *physical* record, which may be larger or smaller.

logical unit (LU)

1. A name that is used in programming to represent an I/O device address. *physical unit (PU), system services control point (SSCP), primary logical unit (PLU), and secondary logical unit (SLU).*
2. In SNA, a port through which a user accesses the SNA network,
   a. To communicate with another user and
   b. To access the functions of the SSCP. An LU can support at least two sessions. One with an SSCP and one with another LU and might be capable of supporting many sessions with other LUs.

logical unit name

In programming, a name that is used to represent the address of an input/output unit.

logical unit 6.2

A SNA/SDLC protocol for communication between programs in a distributed processing environment. LU 6.2 is characterized by

1. A peer relationship between session partners,
2. Efficient utilization of a session for multiple transactions,
3. Comprehensive end-to-end error processing, and
4. A generic Application Programming Interface (API) consisting of structured verbs that are mapped into a product implementation.

logons interpret interpret routine

In VTAM, an installation exit routine, which is associated with an interpret table entry, that translates logon information. It also verifies the logon.

LPAR mode

Logically partitioned mode. The CP mode that is available on the Configuration (CONFIG) frame when the PR/SM feature is installed. LPAR mode allows the operator to allocate the hardware resources of the processor unit among several logical partitions.

M

macro definition

A set of statements and instructions that defines the name of, format of, and conditions for generating a sequence of assembler statements and machine instructions from a single source statement.

macro expansion

See *macro generation*

macro generation

An assembler operation by which a macro instruction gets replaced in the program by the statements of its definition. It takes place before assembly. Synonymous with *macro expansion*.

macro (instruction)

1. In assembler programming, a user-invented assembler statement that causes the assembler to process a set of statements that are defined previously in the macro definition.

2. A sequence of VSE/ICCF commands that are defined to cause a sequence of certain actions to be performed in response to one request.

maintain system history program (MSHP)

A program that is used for automating and controlling various installation, tailoring, and service activities for a VSE system.

main task

The main program within a partition in a multiprogramming environment.

master console

In z/VSE, one or more consoles that receive all system messages, except for those that are directed to one particular console. Contrast this with the *user* console, which receives only those messages that are specifically directed to it, for example messages that are issued from a job that was submitted with the request to echo its messages to that console. The operator of a master console can reply to all outstanding messages and enter all system commands.

maximum (max) CA

A unit of allocation equivalent to the maximum control area size on a count-key-data or fixed-block device. On a CKD device, the max CA is equal to one cylinder.

memory object

Chunk of virtual storage that is allocated above the bar (2 GB) to be created with the IARV64 macro.

message

In VSE, a communication that is sent from a program to the operator or user. It can appear on a console, a display terminal or on a printout.

MSHP

See maintain system history program.

multitasking

Concurrent running of one main task and one or several subtasks in the same partition.

MVS

Multiple Virtual Storage. Implies MVS/390, MVS/XA, MVS/ESA, and the MVS element of the z/OS (OS/390) operating system.

N

NetView

A z/VSE optional program that is used to monitor a network, manage it, and diagnose its problems.

network address

In SNA, an address, consisting of subarea and element fields, that identifies a link, link station, or NAU. Subarea nodes use network addresses; peripheral nodes use local addresses. The boundary function in the subarea node to which a peripheral node is attached transforms local addresses to network addresses and vice versa. See also *network name*.

network addressable unit (NAU)

In SNA, a logical unit, a physical unit, or a system services control point. It is the origin or the destination of information that is transmitted by the path control network. Each NAU has a network address that represents it to the path control network. See also *network name, network address*.

Network Control Program (NCP)

An IBM licensed program that provides communication controller support for single-domain, multiple-domain, and interconnected network capability. Its full name is ACF/NCP.

network definition table (NDT)

In VSE/POWER networking, the table where every node in the network is listed.

network name

1. In SNA, the symbolic identifier by which users refer to a NAU, link, or link station. See also *network address*.
2. In a multiple-domain network, the name of the APPL statement defining a VTAM application program. This is its network name, which must be unique across domains.

node

1. In SNA, an end point of a link or junction common to several links in a network. Nodes can be distributed to host processors, communication controllers, cluster controllers, or terminals. Nodes can vary in routing and other functional capabilities.
2. In VTAM, a point in a network that is defined by a symbolic name. Synonymous with *network node*. See *major node and minor node*.

node type

In SNA, a designation of a node according to the protocols it supports and the network addressable units (NAUs) it can contain.

O

object module (program)

A program unit that is the output of an assembler or compiler and is input to a linkage editor.

online application program

An interactive program that is used at display stations. When active, it waits for data. Once input arrives, it processes it and send a response to the display station or to another device.

operator command

A statement to a control program, issued via a console or terminal. It causes the control program to provide requested information, alter normal operations, initiate new operations, or end existing operations.

optional licensed program

An IBM licensed program that a user can install on VSE by way of available installation-assist support.

output parameter text block (OPTB)

in VSE/POWER's spool-access support, information that is contained in an output queue record if a * $$ LST or * $$ PUN statement includes any user-defined keywords that have been defined for autostart.

P

page data set (PDS)

One or more extents of disk storage in which pages are stored when they are not needed in processor storage.

page fixing

Marking a page so that it is held in processor storage until explicitly released. Until then, it cannot be paged out.

page I/O

Page-in and page-out operations.

page pool

The set of page frames available for paging virtual-mode programs.

panel

The complete set of information that is shown in a single display on terminal screen. Scrolling back and forth through panels like turning manual pages. See also *selection panel*.

partition balancing

A z/VSE facility that allows the user to specify that two or more or all partitions of the system should receive about the same amount of time on the processor.

PASN-AL (primary address space number - access list)

The access list that is associated with a partition. A program uses the PASN-AL associated with its partition and the DU-AL associated with its task (work unit). See also *DU-AL*.

Each partition has its own unique PASN-AL. All programs running in this partition can access data spaces through the PASN-AL. Thus a program can create a data space, add an entry for it in the PASN-AL, and obtain the ALET that indexes the entry. By passing the ALET to other programs in the partition, the program can share the data space with other programs running in the same partition.

PDS

Page data sets.

phase

The smallest complete unit of executable code that can be loaded into virtual storage.

physical record

The amount of data that is transferred to or from auxiliary storage. Synonymous with *block*.

PNET

Programming support available with VSE/POWER; it provides for the transmission of selected jobs, operator commands, messages, and program output between the nodes of a network.

POWER

See *VSE/POWER*.

pregenerated operating system

An operating system such as z/VSE that is shipped by IBM mainly in object code. IBM defines such key characteristics as the size of the main control program, the organization, and size of libraries, and required system areas on disk. The customer does not have to generate an operating system.

preventive service

The installation of one or more PTFs on a VSE system to avoid the occurrence of anticipated problems.

primary address space

In z/VSE, the address space where a partition is executed. A program in primary mode fetches data from the primary address space.

primary library

A VSE library owned and directly accessible by a certain terminal user.

printer/keyboard mode

Refers to 1050 or 3215 console mode (device dependent).

Print Services Facility (PSF)/VSE

An access method that provides support for the advanced function printers.

private area

The virtual space between the shared area (24 bit) and shared area (31 bit), where (private) partitions are allocated. Its maximum size can be defined during IPL. See also *shared area*.

private memory object

Memory object (chunk of virtual storage) that is allocated above the 2 GB line (bar) only accessible by the partition that created it.

private partition

Any of the system's partitions that are not defined as shared. See also *shared partition*.

production library

1. In a pre-generated operating system (or product), the program library that contains the object code for this system (or product).
2. A library that contains data that is needed for normal processing. Contrast with *test library*.

programmer logical unit

A logical unit available primarily for user-written programs. See also *logical unit name*.

program temporary fix (PTF)

A solution or by-pass of one or more problems that are documented in APARs. PTFs are distributed to IBM customers for preventive service to a current release of a program.

PSF/VSE

Print Services Facility/VSE.

PTF

See *Program temporary fix*.

Q

Queue Control Area (QCA)

In VSE/POWER, an area of the data file, which might contain:

- Extended checkpoint information
- Control information for a shared environment.

queue file

A direct-access file that is maintained by VSE/POWER that holds control information for the spooling of job input and job output.

R

random processing

The treatment of data without respect to its location on disk storage, and in an arbitrary sequence that is governed by the input against which it is to be processed.

real address area

In z/VSE, processor storage to be accessed with dynamic address translation (DAT) off

real address space

The address space whose addresses map one-to-one to the addresses in processor storage.

real mode

In VSE, a processing mode in which a program might not be paged. Contrast with *virtual mode*.

recovery management support (RMS)

System routines that gather information about hardware failures and that initiate a retry of an operation that failed because of processor, I/O device, or channel errors.

refresh release

An upgraded VSE system with the latest level of maintenance for a release.

relative-record file

A VSE/VSAM file whose records are loaded into fixed-length slots and accessed by the relative-record numbers of these slots.

release upgrade

Use of the FSU functions to install a new release of z/VSE.

relocatable module

A library member of the type object. It consists of one or more control sections cataloged as one member.

relocating loader

A function that modifies addresses of a phase, if necessary, and loads the phase for running into the partition that is selected by the user.

remote interface

In the context of z/VSE, the remote interface allows a client to make method calls to an EJB although the EJB is on a remote z/VSE host. The container uses the remote interface to create client-side stubs and server-side proxy objects to handle incoming method calls from a client to an EJB.

remote procedure call (RPC)

1. A facility that a client uses to request the execution of a procedure call from a server. This facility includes a library of procedures and an external data representation.
2. A client request to service provider in another node.

residency mode (RMODE)

A program attribute that refers to the location where a program is expected to reside in virtual storage. RMODE 24 indicates that the program must reside in the 24-bit addressable area (below 16 megabytes), RMODE ANY indicates that the program can reside anywhere in 31-bit addressable storage (above or below 16 megabytes).

REXX/VSE

A general-purpose programming language, which is particularly suitable for command procedures, rapid batch program development, prototyping, and personal utilities.

RMS

Recovery management support.

RPG II

A commercially oriented programming language that is specifically designed for writing application programs that are intended for business data processing.

S

SAM ESDS file

A SAM file that is managed in VSE/VSAM space, so it can be accessed by both SAM and VSE/VSAM macros.

SCP

System control programming.

SDL

System directory list.

search chain

The order in which chained sublibraries are searched for the retrieval of a certain library member of a specified type.

second-level directory

A table in the SVA containing the highest phase names that are found on the directory tracks of the system sublibrary.

Secure Sockets Layer (SSL)

A security protocol that allows the client to authenticate the server and all data and requests to be encrypted. SSL was developed by Netscape Communications Corp. and RSA Data Security, Inc..

segmentation

In VSE/POWER, a facility that breaks list or punch output of a program into segments so that printing or punching can start before this program has finished generating such output.

selection panel

A displayed list of items from which a user can make a selection. Synonymous with *menu*.

sense

Determine, on request or automatically, the status or the characteristics of a certain I/O or communication device.

sequential access method (SAM)

A data access method that writes to and reads from an I/O device record after record (or block after block). On request, the support performs device control operations such as line spacing or page ejects on a printer or skip some tape marks on a tape drive.

service node

Within the VSE unattended node support, a processor that is used to install and test a master VSE system, which is copied for distribution to the unattended nodes. Also, program fixes are first applied at the service node and then sent to the unattended nodes.

service program

A computer program that performs function in support of the system. See with *utility program*.

service refresh

A form of service containing the current version of all software. Also referred to as a *system refresh*.

service unit

One or more PTFs on disk or tape (cartridge).

shared area

In z/VSE, shared areas (24 bit) contain the Supervisor areas and SVA (24 bit) and shared areas (31 bit) the SVA (31 bit). Shared areas (24 bit) are at the beginning of the address space (below 16 MB), shared area (31 bit) at the end (below 2 GB).

shared disk option

An option that lets independent computer systems use common data on shared disk devices.

shared memory objects

Chunks of virtual storage allocated above the 2 GB line (bar), that can be shared among partitions.

shared partition

In z/VSE, a partition that is allocated for a program (VSE/POWER, for example) that provides services and communicates with programs in other partitions of the system's virtual address spaces. In most cases shared partitions are no longer required.

shared spooling

A function that permits the VSE/POWER account file, data file, and queue file to be shared among several computer systems with VSE/POWER.

shared virtual area (SVA)

In z/VSE, a high address area that contains a list system directory list (SDL) of frequently used phases, resident programs that are shared between partitions, and an area for system support.

SIT (System Initialization Table)

A table in CICS that contains data used the system initialization process. In particular, the SIT can identify (by suffix characters) the version of CICS system control programs and CICS tables that you have specified and that are to be loaded.

skeleton

A set of control statements, instructions, or both, that requires user-specific information to be inserted before it can be submitted for processing.

socksified

See *socks-enabled*.

Socks-enabled

Pertaining to TCP/IP software, or to a specific TCP/IP application, that understands the *socks protocol*. "Socksified" is a slang term for socks-enabled.

socks protocol

A protocol that enables an application in a secure network to communicate through a firewall via a *socks server*.

socks server

A circuit-level gateway that provides a secure one-way connection through a firewall to server applications in a nonsecure network.

source member

A library member containing source statements in any of the programming languages that are supported by VSE.

split

To double a specific unit of storage space (CI or CA) dynamically when the specified minimum of free space gets used up by new records.

spooling

The use of disk storage as buffer storage to reduce processing delays when transferring data between peripheral equipment and the processor of a computer. In z/VSE, this is done under the control of VSE/POWER.

Spool Access Protection

An optional feature of VSE/POWER that restricts individual spool file entry access to user IDs that have been authenticated by having performed a security logon.

spool file

1. A file that contains output data that is saved for later processing.
2. One of three VSE/POWER files on disk: queue file, data file, and account file.

SSL

See Secure Sockets Layer.

stacked tape

An IBM supplied product-shipment tape containing the code of several licensed programs.

standard label

A fixed-format record that identifies a volume of data such as a tape reel or a file that is part of a volume of data.

stand-alone program

A program that runs independently of (not controlled by) the VSE system.

startup

The process of performing IPL of the operating system and of getting all subsystems and applications programs ready for operation.

start option

In VTAM, a user-specified or IBM specified option that determines conditions for the time a VTAM system is operating. Start options can be predefined or specified when VTAM is started.

static partition

A partition, which is defined at IPL time and occupying a defined amount of virtual storage that remains constant. See also *dynamic partition*.

storage director

An independent component of a storage control unit; it performs all of the functions of a storage control unit and thus provides one access path to the disk devices that are attached to it. A storage control unit has two storage directors.

storage fragmentation

Inability to allocate unused sections (fragments) of storage in the real or virtual address range of virtual storage.

suballocated file

A VSE/VSAM file that occupies a portion of an already defined data space. The data space might contain other files. See also *unique file*.

sublibrary

In VSE, a subdivision of a library. Members can only be accessed in a sublibrary.

sublibrary directory

An index for the system to locate a member in the accessed sublibrary.

submit

A VSE/POWER function that passes a job to the system for processing.

SVA

See shared virtual area.

Synchronous DataLink Control (SDLC)

A discipline for managing synchronous, code-transparent, serial-by-bit information transfer over a link connection. Transmission exchanges might be duplex or half-duplex over switched or non-switched links. The configuration of the link connection might be point-to-point, multipoint, or loop.

SYSRES

See system residence volume.

system control programming (SCP)

IBM supplied, non-licensed program fundamental to the operation of a system or to its service or both.

system directory list (SDL)

A list containing directory entries of frequently used phases and of all phases resident in the SVA. The list resides in the SVA.

system file

In z/VSE, a file that is used by the operating system, for example, the hardcopy file, the recorder file, the page data set.

System Initialization Table (SIT)

A table in CICS that contains data that is used by the system initialization process. In particular, the SIT can identify (by suffix characters) the version of CICS system control programs and CICS tables that you have specified and that are to be loaded.

system recorder file

The file that is used to record hardware reliability data. Synonymous with *recorder file*.

system refresh

See *service refresh*.

system refresh release

See *refresh release*.

system residence file (SYSRES)

The z/VSE system sublibrary IJSYSRS.SYSLIB that contains the operating system. It is stored on the system residence volume DORSES.

system residence volume (SYSRES)

The disk volume on which the system sublibrary is stored and from which the hardware retrieves the initial program load routine for system startup.

system sublibrary

The sublibrary that contains the operating system. It is stored on the system residence volume (SYSRES).

T

task management

The functions of a control program that control the use, by tasks, of the processor and other resources (except for input/output devices).

time event scheduling support

In VSE/POWER, the time event scheduling support offers the possibility to schedule jobs for processing in a partition at a predefined time once repetitively. The time event scheduling operands of the * $$ JOB statement are used to specify the wanted scheduling time.

TLS

See Transport Layer Security.

track group

In VSE/POWER, the basic organizational unit of a file for CKD devices.

track hold

A function that protects a track that is being updated by one program from being accessed by another program.

transaction

1. In a batch or remote batch entry, a job or job step. 2. In CICS TS, one or more application programs that can be used by a display station operator. A given transaction can be used concurrently from one or more display stations. The execution of a transaction for a certain operator is also referred to as a task.
2. A given task can relate only to one operator.

transient area

An area within the control program that is used to provide high-priority system services on demand.

Transport Layer Security

The newest SSL cryptographic protocol. It provides additional strength to privacy and data integrity.

Turbo Dispatcher

A facility of z/VSE that allows to use multiprocessor systems (also called CEC: Central Electronic Complexes). Each CPU within such a CEC has accesses to be shared virtual areas of z/VSE: supervisor, shared areas (24 bit), and shared areas (31 bit). The CPUs have equal rights, which means that any CPU might receive interrupts and work units are not dedicated to any specific CPU.

U

UCB

Universal character set buffer.

universal character set buffer (UCB)

A buffer to hold UCS information.

UCS

Universal character set.

user console

In z/VSE, a console that receives only those system messages that are specifically directed to it. These are, for example, messages that are issued from a job that was submitted with the request to echo its messages to that console. Contrast with *master console*.

user exit

A programming service that is provided by an IBM software product that can be requested during the execution of an application program for the service of transferring control back to the application program upon the later occurrence of a user-specified event.

V

variable-length relative-record data set (VRDS)

A relative-record data set with variable-length records. See also *relative-record data set*.

variable-length relative-record file

A VSE/VSAM relative-record file with variable-length records. See also *relative-record file*.

VIO

See virtual I/O area.

virtual address

An address that refers to a location in virtual storage. It is translated by the system to a processor storage address when the information stored at the virtual address is to be used.

virtual addressability extension (VAE)

A storage management support that allows to use multiple virtual address spaces.

virtual address space

A subdivision of the virtual address area (virtual storage) available to the user for the allocation of private, nonshared partitions.

virtual disk

A range of up to 2 gigabytes of contiguous virtual storage addresses that a program can use as workspace. Although the virtual disk exists in storage, it appears as a real FBA disk device to the user program. All I/O operations that are directed to a virtual disk are intercepted and the data to be written to, or read from, the disk is moved to or from a data space.

Like a data space, a virtual disk can hold only user data; it does not contain shared areas, system data, or programs. Unlike an address space or a data space, data is not directly addressable on a virtual disk. To manipulate data on a virtual disk, the program must perform I/O operations.

Starting with z/VSE 5.2, a virtual disk may be defined in a shared memory object.

virtual I/O area (VIO)

An extension of the page data set; used by the system as intermediate storage, primarily for control data.

virtual mode

The operating mode of a program, where the virtual storage of the program can be paged, if not enough processor (real) storage is available to back the virtual storage.

virtual partition

In VSE, a division of the dynamic area of virtual storage.

virtual storage

Addressable space image for the user from which instructions and data are mapped into processor storage locations.

virtual tape

In z/VSE, a virtual tape is a file (or data set) containing a tape image. You can read from or write to a virtual tape in the same way as if it were a physical tape. A virtual tape can be:

- A VSE/VSAM ESDS file on the z/VSE local system.
- A remote file on the server side; for example, a Linux, UNIX, or Windows file. To access such a remote virtual tape, a TCP/IP connection is required between z/VSE and the remote system.

volume ID

The volume serial number, which is a number in a volume label that is assigned when a volume is prepared for use by the system.

VRDS

Variable-length relative-record data sets. See *variable-length relative record file*.

VSAM

See *VSE/VSAM*.

VSE (Virtual Storage Extended)

A system that consists of a basic operating system and any IBM supplied and user-written programs that are required to meet the data processing needs of a user. VSE and hardware it controls form a complete computing system. Its current version is called z/VSE.

VSE/Advanced Functions

A program that provides basic system control and includes the supervisor and system programs such as the Librarian and the Linkage Editor.

VSE Connector Server

Is the host part of the VSE JavaBeans, and is started using the job STARTVCS, which is placed in the reader queue during installation of z/VSE. Runs by default in dynamic class R.

VSE/DITTO (VSE/Data Interfile Transfer, Testing, and Operations Utility)

An IBM licensed program that provides file-to-file services for disk, tape, and card devices.

VSE/ESA (Virtual Storage Extended/Enterprise Systems Architecture)

The predecessor system of z/VSE.

VSE/Fast Copy

A utility program for fast copy data operations from disk to disk and dump/restore operations via an intermediate dump file on magnetic tape or disk.

VSE/FCOPY (VSE/Fast Copy Data Set program)

An IBM licensed program for fast copy data operations from disk to disk and dump/restore operations via an intermediate dump file on magnetic tape or disk. There is also a stand-alone version: the FASTCOPY utility.

VSE/ICCF (VSE/Interactive Computing and Control Facility)

An IBM licensed program that serves as interface, on a time-slice basis, to authorized users of terminals that are linked to the system's processor.

VSE/ICCF library

A file that is composed of smaller files (libraries) including system and user data, which can be accessed under the control of VSE/ICCF.

VSE JavaBeans

Are JavaBeans that allow access to all VSE-based file systems (VSE/VSAM, Librarian, and VSE/ICCF), submit jobs, and access the z/VSE operator console. The class library is contained in the *VSEConnector.jar* archive. See also *JavaBeans*.

VSE library

A collection of programs in various forms and storage dumps stored on disk. The form of a program is indicated by its member type such as source code, object module, phase, or procedure. A VSE library consists of at least one sublibrary, which can contain any type of member.

VSE/POWER

An IBM licensed program that is primarily used to spool input and output. The program's networking functions enable a VSE system to exchange files with or run jobs on another remote processor.

VSE/VSAM (VSE/Virtual Storage Access Method)

An IBM access method for direct or sequential processing of fixed and variable length records on disk devices.

VSE/VSAM catalog

A file containing extensive file and volume information that VSE/VSAM requires to locate files, to allocate and deallocate storage space, to verify the authorization of a program or an operator to gain access to a file, and to accumulate use statistics for files.

VSE/VSAM managed space

A user-defined space on disk that is placed under the control of VSE/VSAM.

W

wait for run subqueue

In VSE/POWER, a subqueue of the reader queue with dispatchable jobs ordered in execution start time sequence.

wait state

The condition of a processor when all operations are suspended. System recovery from a hard wait is impossible without performing a new system startup. See *hard wait*.

Workstation File Transfer Support

Enables the exchange of data between IBM Personal Computers (PCs) linked to a z/VSE host system where the data is kept in intermediate storage. PC users can retrieve that data and work with it independently of z/VSE.

work file

A file that is used for temporary storage of data being processed.

Numerics

24-bit addressing

Provides addressability for address spaces up to 16 megabytes.

31-bit addressing

Provides addressability for address spaces up to 2 gigabytes.

64-bit addressing

Provides addressability for address spaces up to 2 gigabytes and above.

Index

Product Number: 5609-ZV5

SC34-2628-03