English

FUJITSU Software

BS2000 OSD/BC V11.0

Diagnostics Handbook

User Guide

Edition July 2017

Comments… Suggestions… Corrections…

The User Documentation Department would like to know your opinion on this manual. Your feedback helps us to optimize our documentation to suit your individual needs.

Feel free to send us your comments by e-mail to:
manuals@ts.fujitsu.com

Certified documentation according to DIN EN ISO 9001:2008

To ensure a consistently high quality standard and user-friendliness, this documentation was created to meet the regulations of a quality management system which complies with the requirements of the standard DIN EN ISO 9001:2008.

cognitas. Gesellschaft für Technik-Dokumentation mbH
www.cognitas.de

Copyright and Trademarks

Contents

1 Preface

Any user wishing to get to the bottom of problems affecting the running of the operating system or the execution of application programs must have access to information on the status of the operating system and/or application program at the time of the error and for the short period preceding the error.

This “Diagnostics Handbook” describes the facilities provided in BS2000 which enable customers using this operating system to obtain and evaluate information about any software errors which crop up and other program execution data.

1.1 Objectives and target groups of this manual

The “Diagnostics Handbook” is intended for system programmers already acquainted with the system, systems support staff and software service staff. It describes the software products and components which are important for diagnosis.

This manual is intended for both privileged and nonprivileged BS2000 users.

1.2 Summary of contents

This manual describes the software diagnosis methods in BS2000.

The utility routines provided in BS2000 are described which enable information regarding software errors which occur and any other program execution data to be obtained and evaluated.

The only such facility not described in this manual is the product AID. AID is dealt with in the “AID” manual [1].

Readme file

The functional changes to the current product version and revisions to this manual are described in the product-specific Readme file.

Readme files are available to you online in addition to the product manuals under the various products at *http://manuals.ts.fujitsu.com*. You will also find the Readme files on the Softbook DVD.

Information under BS2000

When a Readme file exists for a product version, you will find the following file on the BS2000 system:

`SYSRME.<product>.<version>.<lang>`

This file contains brief information on the Readme file in English or German (<lang>=E/D). You can view this information on screen using the `/SHOW-FILE` command or an editor. The `/SHOW-INSTALLATION-PATH INSTALLATION-UNIT=<product>` command shows the user ID under which the product's files are stored.

Additional product information

Current information, version and hardware dependencies, and instructions for installing and using a product version are contained in the associated Release Notice. These Release Notices are available online at *http://manuals.ts.fujitsu.com*.

1.3 Changes since the last edition of the manual

The following important changes have been introduced in this edition of the "Diagnostics Handbook" since the publication of the previous edition:

- The manual has been adapted for BS2000 OSD/BC V11.0.
- All references to SQ servers, which are no longer supported, have been removed.
- Chapter on DAMP
  - Maximum number of PAM pages of a file increased to X'1FE00000' and maximum number of segments of a file increased to X'FE'.
  - Online help is also available in English.
  - New statement COMPILE-PRODAMP-PROCEDURE.
- Chapter on SLED: a SLED file may be larger than 32 GB (max. 256 GB) and may be located on the home pubset.

1.4 Notational conventions

The metasyntax used in this manual to describe statements is explained in the manual “Commands” [8]. Operands reserved for privileged users are shown in the formats with a gray background.

Because the names are frequently mentioned, for the sake of simplicity and clarity the following abbreviations are used:

- **BS2000 servers** for the servers with /390 architecture and the servers with x86 architecture. These servers are operated with the corresponding BS2000 operating system.
- **/390 servers** for the Server Unit /390 of the FUJITSU Server BS2000 SE Series and the Business Servers of the S Series
- **x86 servers** for the Server Unit x86 of the FUJITSU Server BS2000 SE Series
- **SE servers** for the FUJITSU Server BS2000 SE Series (Server Units /390 and x86)
- **S servers** for the Business Servers of the S Series (/390 architecture)

In the examples the strings <date>, <time> and <version> specify the current outputs for date, time and version of a software product when the examples are otherwise independent of the date, time and version.

The following typographical elements are used in this manual:

For notes on particularly important information

This symbol designates special information that points out the possibility that data can be lost or that other serious damage may occur.

[ ] References to other publications within the text are given in abbreviated form followed by numbers; the full titles are listed in the “References” section at the back of this manual.

`input` Inputs and system outputs in examples are shown in typewriter font

2 Software diagnosis methods in BS2000

Information on program execution can be collected in two different ways: by logging the progress of execution and by dumping the contents of the memory at the time of an actual or suspected error. Both approaches, and how they are implemented in BS2000, are described in the following sections.

Command	Meaning
ACTIVATE-SNAPSHOT	Activate dump generator SNAP
ADD-ASE-ELEMENT	Declare ASE element
ADD-USER	Permit user-specific control of the hardware or linkage AUDIT, define user-specific test privilege
CANCEL-JOB	Control of CDUMP dump output and other purposes
CHANGE-SERSLOG-FILE	Closes the current SERSLOG file and opens a new one
CREATE-DUMP	Request a user or system dump
DEACTIVATE-SNAPSHOT	Deactivate dump generator SNAP
EXIT-JOB	Control of CDUMP dump output and other purposes
FORCE-CANCEL-JOB	Control of CDUMP dump output and other purposes
HOLD-HARDWARE-AUDIT	Interrupt hardware AUDIT mode
HOLD-LINKAGE-AUDIT	Interrupt linkage AUDIT mode
MODIFY-ASE-PARAMETERS	Modify global ASE settings
MODIFY-FILE-ATTRIBUTES	Control access to shareable files on a file-specific basis for a user ID with the HARDWARE-MAINTENANCE privilege
MODIFY-TEST-OPTIONS	Modify AID test privilege values locally and the control of dump requests, permit task-local hardware or linkage AUDIT. Control CDUMP output Control the AID tests of other tasks under your own user ID with low testing privileges
MODIFY-USER-ATTRIBUTES	Permit user-specific control of the hardware or linkage AUDIT. Modify test privileges user-by-user
REMOVE-ASE-ELEMENT	Delete ASE element
RESUME-HARDWARE-AUDIT	Resume interrupted hardware AUDIT mode

Table 1: Overview of interfaces for diagnosis methods in BS2000 (part 1 of 2)

Command	Meaning
RESUME-LINKAGE-AUDIT	Resume interrupted linkage AUDIT mode
SHOW-ASE-ELEMENT	Display ASE element
SHOW-ASE-LOGGING	Display data of internal ASE logging
SHOW-ASE-PARAMETERS	Display global ASE settings
SHOW-ASE-STATUS	Display ASE status information
SHOW-AUDIT-STATUS	Shows the status of the linkage AUDIT and the hardware AUDIT
SHOW-HARDWARE-AUDIT	Request output of the hardware AUDIT table to SYSOUT or SYSLST
SHOW-LINKAGE-AUDIT	Request output of the linkage AUDIT table to SYSOUT or SYSLST
SHOW-SERSLOG-STATUS	Shows the status of SERSLOG
SHOW-SNAPSHOT-STATUS	Display information on SNAP dump
SHOW-TEST-OPTIONS	Display task-specific settings for test and diagnosis
SHOW-TRACE-STATUS	Request attribute and status information on system traces
START-DAMP	Start DAMP
START-ELFE	Start ELFE
START-HARDWARE-AUDIT	Start hardware AUDIT mode
START-LINKAGE-AUDIT	Start linkage AUDIT mode
START-NDM-DIAGNOSIS	Start NDMDUMP
START-SERSLOG	Activate SERSLOG
START-TRACE	Activate switchable trace
STOP-HARDWARE-AUDIT	Terminate hardware AUDIT mode and release the AUDIT table
STOP-LINKAGE-AUDIT	Terminate linkage AUDIT mode and release the AUDIT table
STOP-SERSLOG	Deactivate SERSLOG
STOP-TRACE	Switch off active trace
Macro	**Meaning**
AUDIT	Use hardware and linkage AUDIT functions
BKPT	Transfer control to the system
CDUMP2	Initiate a memory dump without terminating the program
TERM	Terminate program and job section; if necessary, initiate a memory dump

Table 1: Overview of interfaces for diagnosis methods in BS2000 (part 2 of 2)

2.1 Logging the progress of software execution

Logging the progress of software execution is a prophylactic measure designed to provide information on the causes of any software error which may occur. Certain information is stored in such a log continuously, i.e. even during normal (error-free) program execution. This information may take the form of memory addresses, the names of any modules which are used, program line numbers, etc. If an error occurs, it may be possible to localize it with the aid of the log. However, since this method results in the rapid accumulation of a large quantity of data (not all of which is necessarily useful for error analysis), it is sound practice to discard the data after a certain time. This can be done, for example, by overwriting the existing data cyclically, retaining only the last n entries in the log; in the event of an error, this is sufficient to ensure that the steps which led to the error are available for analysis.

The logging methods used in BS2000 are called error logging and tracing.

2.1.1 Software error logging

“Software error logging” is a method which stores error entries (error records) in a logging file whenever a software problem is detected. The software module which is active at the time of the error initiates the logging file entry and also defines the contents of such an error record.

In BS2000, the generation and writing of error entries are handled by the components ELS and SERSLOG. ELS logs all hardware faults and is described in the “ELSA” manual [3]. Software errors are recorded by SERSLOG, which is described in the present manual as of chapter “SERSLOG Software error logging in the SERSLOG file” on page 363.

2.1.2 Tracing

“Tracing” denotes the logging of execution or selected parts thereof. This involves the continuous recording of certain data, regardless of whether or not a software problem actually exists. This data is then retained for a certain time, i.e. until it is overwritten by newer data. The recorded logs are managed by TRACE MANAGER (see chapter “TRACE MANAGER Collect diagnostic information during the session” on page 401).

During normal system operation, a number of traces are logged automatically, and additional ones can be activated as required in critical cases. However, some of these additional traces can be very time-consuming.

The AUDIT function (see chapter “AUDIT Log addresses of executed branch instructions” on page 21) constitutes a special kind of tracing. AUDIT records all branch instructions whose conditions are fulfilled. This function is also available to nonprivileged users.

The activation of additional traces or of the system-wide AUDIT can severely impair the performance of the operating system and should therefore be avoided unless absolutely necessary.

2.2 Saving the contents of memory

A memory dump saves the contents of, for example, registers, real memory, address spaces or files at the time of an actual or suspected error. A dump is started either automatically by an error detection facility of the operating system or of the currently active program or explicitly by entering an appropriate command at the terminal or console. Depending on the seriousness or scope of the error which has occurred, a distinction is made in BS2000 between:

- a full dump, which also terminates the operating system program, and
- a partial dump, which does not terminate the operating system.

If the cause and effect of an error cannot be localized, and if an essential part of the operating system is affected, all main memory and background memory areas of BS2000 must be saved for diagnosis. Dumps of this sort are generated by the program SLED (**S**elf-**L**oading **E**mergency **D**ump, see chapter “SLED dump” on page 367). SLED operates independently of BS2000.
Once a full dump has been generated, the operating system program is reloaded. It is possible to set up the system so that saving of the memory contents and restarting of the operating system are carried out automatically.

If the error can be narrowed down to a single task, only that part of the memory contents used by the task needs to be saved. If the error occurred in the user program, a partial dump therefore tends to cover solely the user address space of the errored task. However, if the error occurred in an operating system function and has no consequences for other tasks, then parts of the operating system address space are also saved.

Whereas the errored task is normally terminated by the error handling routine after a partial dump has been generated, the operating system and all other tasks continue to run both during and after output of the dump.

In BS2000, partial dumps (system, area or user dumps) can be created by means of the CDUMP2 macro (see chapter “CDUMP Output area, user or system dump” on page 27). CDUMP2 runs asynchronously, i.e. concurrently with other tasks under BS2000 control. The scope of the partial dump generated by way of CDUMP2 varies depending on the specified operands and on the privilege of the calling task.

SNAP dumps are a special form of partial dump (see chapter “SNAP dump” on page 395). SNAP saves the class 1 and class 3 memory of the operating system and the name and entry point lists of all operating system modules (EOLDTAB).

SNAP is called by the operating system from the status TPR or SIH. This generally happens when an unusual operating system state arises that is not serious enough to terminate the session. Following the call, SNAP automatically suspends BS2000 operation for a maximum of 24 seconds, generates a dump independently of BS2000, and then reactivates the operating system.

2.3 Evaluating dumps and logging data

The data saved by a dump or in a log can be evaluated later at any time.

The error logging file created by SERSLOG is evaluated by the program ELFE (see chapter “ELFE Edit and evaluate the SERSLOG file” on page 351), which is capable of sorting the individual entries on the basis of various criteria and then outputting them. ELFE also provides statistical functions (such as recording how often each error has occurred, etc.).

Traces are not written to files, but are kept in main memory and cannot be output separately. They are, however, also saved on generating a dump and are therefore contained in the dump file and can be output when evaluating the dump. AUDIT traces are an exception. They can be output interactively with the SHOW commands of the AUDIT function.

A SLED dump can be analyzed with DAMP.
DAMP (see chapter “DAMP Dump analysis” on page 47), by contrast, is an interactive editing routine, i.e. the actual diagnosis is performed on the screen. This enables the interactive user to track links between related processing steps or display tables on the screen and then initiate any further diagnostic steps which prove necessary.
Partial dumps generated using SNAP or CDUMP (system, user or area dumps) can also be evaluated interactively on the screen with the aid of DAMP.

The actual diagnosis presupposes familiarity with the system.

2.4 Data privacy

2.4.1 Data privacy during output of a dump

Data privacy precautions can be implemented for the generation and storage of a dump. By marking memory pages as “secret pages”, you can prevent them from being transferred from the system dump, user dump or area dump to the dump files.
In order to do this, the system parameter DUMPSEPA must have been set accordingly at system initialization by using the parameter service (see the manual “Introduction to System Administration” [6]). Pages can be marked as SECRET PAGES via the CSTAT macro (see the “Executive Macros” manual [4]) or $CSTA. This is specially advisable when dealing with sensitive data. You should check whether this data may be of relevance for diagnostic purposes.
Dumps which contain confidential data are stored under special user IDs (SYSUSER, SYSDUMP, SYSSNAP, TSOS); the access rights for these user IDs are assigned by systems support.

2.4.2 Data privacy for dump files

Dump files enjoy the same protection facilities as any other file, namely standard access control (USER-ACCESS/ACCESS), the basic access control list (BACL) and guards. As a result, dump files can be protected against unauthorized access by means of defining and assigning appropriate access rights.

2.4.3 Data privacy for logging files

Data destined for logging, e.g. for SERSLOG error logs, cannot be checked with regard to its confidentiality. For this reason, logging files should be protected against unauthorized access in the same way as dump files (see above).

2.4.4 Data privacy during diagnosis in an active system

Diagnosis in an active system is possible with the software product AID (see the “AID” manual [1]) or with DAMP (see chapter “DAMP Dump analysis” on page 47). The system administrator can influence data privacy by configuring the appropriate system parameters and by assigning test privileges (see the manual “Introduction to System Administration” [6]).

3 AUDIT
Log addresses of executed branch instructions

The AUDIT function permits the logging of the addresses of executed branch instructions so that, if necessary, the execution sequence of the program involved can subsequently be reconstructed. Not only can this be used to log execution of the user's own task, but it is also possible to activate AUDIT in another task which, for example, is in a loop or is running as a batch task.

There are two types of AUDIT functions:

- hardware AUDIT
- linkage AUDIT

In contrast to the hardware AUDIT function, which stores in an AUDIT table the source address for every branch executed, the linkage AUDIT function stores the target address in an AUDIT table each time certain branch or load commands are executed. Separate tables are created for the hardware AUDIT and linkage AUDIT functions. The header line indicates the relevant function when the table is output.
The hardware AUDIT and linkage AUDIT can be activated independently of one another. The branch addresses of a process in the user's own task (e.g. contingency process), the entire run of one task or all the tasks of the current session and, in the case of the linkage AUDIT, the branch addresses of a processor can be logged.

The hardware and linkage AUDITs can be prevented or allowed for a complete session by setting the parameter AUDALLOW=YES/NO in the startup parameter service, but this can only be done for both AUDITs together. Setting the parameter AUDALLOW=NO will also automatically disable any linkage AUDIT that may have been enabled for a local processor.

The AUDIT function is offered at command level (see the following sections and the "Commands" manual [8]) and as a macro (AUDIT, see the "Executive Macros" manual [4]).

Hardware and linkage AUDITs can also be allowed or prevented on a user or task basis via the commands ADD-USER, MODIFY-USER-ATTRIBUTES and MODIFY-TEST-OPTIONS.

See also the “Commands” manual [8].

```
AUDIT=*PARAMETERS(...)

  *PARAMETERS(...)
    | HARDWARE-AUDIT = *UNCHANGED/ *NOT-ALLOWED/ *ALLOWED
    | LINKAGE-AUDIT = *UNCHANGED/ *NOT-ALLOWED/ *ALLOWED
```

The parameter *UNCHANGED is not valid for the ADD-USER command.

Any attempts to call a hardware or linkage AUDIT without the required permission are rejected with either a return code (macro call) or a message (command).

The x86 servers lack the firmware requirements to trace branches in programs, so the hardware AUDIT is not available. The commands HOLD-HARDWARE-AUDIT, RESUME-HARDWARE-AUDIT, SHOW-HARDWARE-AUDIT, START-HARDWARE-AUDIT and STOP-HARDWARE-AUDIT are rejected with the message `IDA0020`. Macro calls for the hardware AUDIT return with error code X'00000000'. Note, however, that facilities to analyze TU programs with address stops and runtime tracing are also offered by the product AID.
See the “AID” manual [1] for details.

3.1 Hardware AUDIT

If the hardware AUDIT function is activated, the source addresses of all branch commands executed are stored in the hardware AUDIT table. This table is 256 bytes long (corresponding to 64 single-word entries) and is overwritten cyclically.

If desired, the user can, on request, have the data from the AUDIT table placed in a save table before it is overwritten. This additional table has a maximum size of 64 Kbytes for the TU-AUDIT. For the TPR-AUDIT (only for privileged callers under the user ID TSOS), instead of the 256-byte AUDIT table, only a larger 4-KB AUDIT table is created, and this table also serves as the save table.

The save tables and the large-size AUDIT table are overwritten cyclically and can be output using the SHOW-HARDWARE-AUDIT command and by specifying the relevant operands with the AUDIT macro.

Command	**Meaning**
HOLD-HARDWARE-AUDIT	Interrupt hardware AUDIT mode
RESUME-HARDWARE-AUDIT	Resume hardware AUDIT mode
SHOW-AUDIT-STATUS	Shows the status of the linkage AUDIT and the hardware AUDIT
SHOW-HARDWARE-AUDIT	Output hardware AUDIT table to SYSOUT or SYSLST
START-HARDWARE-AUDIT	Start hardware AUDIT mode
STOP-HARDWARE-AUDIT	Terminate hardware AUDIT mode and release hardware AUDIT table
Macro	**Meaning**
AUDIT	Apply hardware and linkage AUDIT functions

Table 2: Interfaces for hardware AUDIT

3.2 Linkage AUDIT

If the linkage AUDIT function is activated, the destination addresses are logged to a 4-KB trace table (linkage AUDIT trace table) whenever the commands BASR, BALR, BASSM and BAKR are executed. This table is overwritten cyclically and can optionally be stored in the case of TU applications in a save table of up to 64 KB (see page 23) in the same way as for the hardware AUDIT. It is not possible to create an additional save table in TPR. Performance is reduced for the processor state for which the linkage AUDIT is activated. In contrast to the hardware AUDIT, the branches logged by the linkage AUDIT occur far more rarely than the conditional branches logged by the hardware AUDIT.

AUDIT control using the parameter service

The linkage AUDIT enables the linkage AUDIT to be activated on a CPU-specific basis for all CPUs or all logical machines of a server configuration. For each CPU, a trace table is created in privileged class 3 memory and retained throughout the entire session.The linkage AUDIT can be enabled in the startup phase via the parameter service.

The linkage AUDIT is controlled by means of the SYSOPT-IPL parameter record, which can be used to define different specifications for system initialization. The parameter that applies to AUDIT is LINKAGE-AUDIT-SCOPE, which is defined as follows within the SYSOPT-IPL record:

```
/BS2000 PARAMS
/BEGIN SYSOPT-IPL
LINKAGE-AUDIT-SCOPE=NO/INTERRUPT-HANDLING/SYSTEM-LEVEL
/EOF
/END-PARAMS
```

Meanings of the operands

LINKAGE-AUDIT-SCOPE = NO
The linkage AUDIT function is not activated (default value).

LINKAGE-AUDIT-SCOPE = INTERRUPT-HANDLING
The linkage AUDIT function is activated for the SIH processor state.

LINKAGE-AUDIT-SCOPE = SYSTEM-LEVEL
The linkage AUDIT function is activated for the TPR and SIH processor states.

In addition to this special AUDIT control, general control of the AUDIT can be achieved via the system parameter AUDALLOW and various BS2000 commands (see page 23 for details).

AUDIT control in the current session

The START-LINKAGE-AUDIT, STOP-LINKAGE-AUDIT and RESUME-LINKAGE-AUDIT commands and the AUDIT macro are available to the system administrator to enable or disable the AUDIT in the current session.

Command	**Meaning**
HOLD-HARDWARE-AUDIT	Interrupt linkage AUDIT mode
RESUME-HARDWARE-AUDIT	Resume linkage AUDIT mode
SHOW-AUDIT-STATUS	Shows the status of the linkage AUDIT and the hardware AUDIT
SHOW-HARDWARE-AUDIT	Output linkage AUDIT table to SYSOUT or SYSLST
START-HARDWARE-AUDIT	Start linkage AUDIT mode
STOP-HARDWARE-AUDIT	Terminate linkage AUDIT mode and release linkage AUDIT table
Macro	**Meaning**
AUDIT	Apply hardware and linkage AUDIT functions

Table 3: Interfaces for linkage AUDIT

4 CDUMP
Output area, user or system dump

The CDUMP2 macro (see the “Executive Macros” manual [4]) generates, under control of a separate task, a memory dump. Depending on the specified operand value, this dump may be an area dump, a user dump or a system dump (see section “Dump forms” on page 29). The macro generates a 31-bit interface. CDUMP2 alone should be used in the future, since development will only be continued for CDUMP2. The old CDUMP macro must still be used to generate a 24-bit interface.

The dumps are stored in unedited form in a PAM file on disk or tape. Dumps cannot be spread out over a number of tapes. Dumps can be copied from tape to disk using the COPY-FILE command (see example on page 34). The structure of the dump file corresponds to the SLEDFILE format. All the dump types generated by CDUMP can be evaluated with the DAMP diagnostic program (see chapter “DAMP Dump analysis” on page 47).

The CREATE-DUMP command (see the “Commands” manual [8]) is used at command level to start a user dump.

Area and user dumps are cataloged under the user ID of the calling task (error task) unless they contain memory pages with special attributes (see page 36) or read-protected data, in which case they are cataloged under the user ID SYSUSER. System dumps are always cataloged under the user ID SYSDUMP. If there is not enough memory space for the user ID of the calling task or the user ID SYSUSER, an error message is output, and generation of the dump is terminated. Memory is automatically extended for the SYSDUMP user ID and output of the dump continued. If space saturation level 5 is reached during this process, only error logging entries are generated instead of the dump.

In batch or procedure mode, user and area dumps are not generated unless the command MODIFY-TEST-OPTIONS DUMP=YES has been specified.

If a system error during dump processing prevents a user dump from being generated, CDUMP attempts to produce a system dump. If this likewise proves abortive, CDUMP generates a SERSLOG entry containing the CDUMP work area with the TTSAV (internal CDUMP data) as well as the TCB (Task Control Block) and the PCB (Program Control Block).

Command	Meaning
ADD-USER	Define user-specific test privilege
CANCEL-JOB	Control of CDUMP dump output
CREATE-DUMP	Request a user or system dump
EXIT-JOB	Control of CDUMP dump output
FORCE-JOB-CANCEL	Control of CDUMP dump output
MODIFY-TEST-OPTIONS	Control of CDUMP dump output
MODIFY-USER-ATTRIBUTES	Modify test privileges user-by-user
SHOW-TEST-OPTIONS	Display task-specific settings for test and diagnosis
SHOW-USER-ATTRIBUTES	Display user-specific settings for test and diagnosis
Macro	**Meaning**
CDUMP2	Initiate a memory dump without terminating the program
TERM	Terminate program and job section; if necessary, initiate a memory dump

Table 4: Interfaces for dump control

4.1 Dump forms

4.1.1 Area dump

An area dump is requested via the operand specification SCOPE=*AREA of the CDUMP2 macro (permitted only from the TU area). It contains the areas of class 5 and 6 memory selected with the call and is stored under the user ID of the caller (exceptions: see section “Influence of page attributes” on page 36).

Some system parameters also influence memory area output (see section “Control by means of system parameters” on page 38).

The DSCTRL operand can be used to address areas in a data space that are to be included in the area dump.

The default setting is MODE=*STD; in this case, area dumps include the following items in addition to the specified areas

– the AIDSYSD module
– the areas containing COMAREA
– P1-PCB
– the TCB.

If MODE=*EXP is specified, area dumps also contain the following items

– the areas containing COMAREA
– the AIDSYSD module
– trace dump list
– TTSAV (internal CDUMP data)
– P1 AUDIT table
– binder/loader metadata
– memory areas with the following tables
  XTV (eXecutive Vector Table)
  SVMT (System Virtual Memory Table)
  UVMT (User Virtual Memory Table)
  TCB (Task Control Block)
  P1-PCBs (Process Control Block)
– pages to which the reference address specified in the PC operand points

File name of the area dump

CDUMP creates the area dump under one of the following IDs:

– under the user ID of the caller, provided he or she is authorized to read all the data output by the dump.

– under the SYSUSER system ID if the dump contains read-protected data (e.g. programs protected with a read password that the user has not included in the task's password table).
Then only the system administrator can permit access to the user, e.g. by means of

```
/MODIFY-FILE-ATTRIBUTES dateiname, -
/   PROTECTION=*PARAMETERS(USER-ACCESS=*ALL-USERS,READ-PASSWORD=readpass)
```

The file name of an area dump has the following structure:

– `:catid:$userid.SYS.ADUMP[.jobname].tsn.i`,
if the file is saved under the caller's user ID

– `:catid:$SYSUSER.SYS.ADUMP[.jobname].tsn.i.userid`,
if the file is saved under the SYSUSER system ID.

where:

catid	is the catalog ID of the public volume set on which the dump was stored.
userid	is the user ID of the caller.
jobname	is the job name, comprising up to eight characters.
tsn	is the four-digit task sequence number of the error task.
i	is the five-digit sequence number of the area dump.

4.1.2 User dump

A user dump is requested via the operand specification SCOPE=*USER of the CDUMP2 macro or via the CREATE-DUMP command. It covers the entire user address space, i.e. class 5 and class 6 memory.
Some system parameters influence memory area output (see section “Control by means of system parameters” on page 38).

Output options using CDUMP2 operands:
– user-specific data spaces with the DS operand
– user-specific DIV windows with the DIV operand
– imported SINIX files with the MMAP operand

If the error task instruction counter points to the system address space, all the pages referenced via general registers and PCs (up to 256 PCBs of the error task) are output, including the preceding five pages and the following five pages.

User dumps also contain the following memory areas:

– the AIDSYSD module
– trace dump list
– TTSAV (internal CDUMP data)
– P1 AUDIT table
– binder/loader metadata
– system trace table
– memory areas containing the following tables
  XTV (eXecutive Vector Table)
  SVMT (System Virtual Memory Table)
  UVMT (User Virtual Memory Table)
  TCB (Task Control Block)
  P1-PCBs (Process Control Block)
– page to which the reference address specified in the PC operand points
– JIT390 administration data

In the event of a user dump, the system trace table is buffered with its contents as they were when CDUMP was called. When the dump is generated, the system trace table is incorporated in the dump (at the position at which the table is located in the system).

File name of a user dump

CDUMP creates the user dump under one of the following IDs:

– under the user ID of the caller, provided he or she is authorized to read all the data output by the dump.

– under the SYSUSER system ID if the dump contains read-protected data (e.g. programs protected with a read password that the user has not included in the task's password table).

  If the dump includes at least one page that is privileged but not "common readable", the user dump is output to the user ID SYSUSER as well.
  If the dump is stored under $SYSUSER, only the system administrator can permit access to the user, e.g. by means of

```
/MODIFY-FILE-ATTRIBUTES dateiname, -
/   PROTECTION=*PARAMETERS(USER-ACCESS=*ALL-USERS,READ-PASSWORD=readpass)
```

The file name of a user dump has the following structure:

– `:catid:$userid.DUMP[.jobname].tsn.i`,
  if the file is saved under the caller's user ID :

– `:catid:$SYSUSER.DUMP[.jobname].tsn.i.userid`,
  if the file is saved under the SYSUSER system ID.

where:

catid	is the catalog ID of the public volume set on which the dump is stored.
userid	is the user ID of the caller.
jobname	is the name of the job, comprising up to eight characters.
tsn	is the four-digit task sequence number of the error task.
i	is the five-digit sequence number of the user dump.

4.1.3 System dump

A system dump is requested via the operand specification SCOPE=*SYSTEM of the CDUMP2 macro and is always cataloged under the user ID SYSDUMP.
It covers the entire class 6, class 5, class 3 and class 1 memory with the exception of those pages declared “secret pages”. Some system parameters influence memory area output (see section “Control by means of system parameters” on page 38).

The areas containing the following tables are automatically output together with the class 5, class 3 and class 1 memory:

– EXVT
– SVMT
– UVMT
– TCB
– PCB stack
– JCB
– TTSAV
– P1-AUDIT

All data pages of class 4 memory are output except “secret pages”.

The system dump also contains those pages of class 4 and class 2 memory which are offset anywhere up to five pages before or after a reference address and the reference page itself. These reference addresses are pointed to by the program counters (PC) in the PCBs and the trace table and by the general-purpose registers in the PCBs and the bourse registers in the PCBs. Memory pages which have the attribute “secret pages” are not output.

A system dump also contains the following:

– the AIDSYSD, EOLDTAB, DMCHD, NSISINF and CLASS2OP modules
– the trace dump list area
– the REPLOG and SERSLOG system files
  (stored in separate dump segments, down to the last-page pointer).

In the CDUMP2 macro, buffering of the class 1, class 3 and the resident class 4 memory can be requested with the SNAP operand (privileged users only) before the actual creation of the dump. This so-called SNAP dump is then incorporated into the dump when generating the system dump.

The operator is issued a message (`IDA0N52`) inquiring as to whether the system dump is to be output to disk or tape. Output of the message can be suppressed with the DUMPCTRL and DUMPSD# system parameters .

Normally, a system dump can be requested only from the privileged system area (TPR). However, you can also request a system dump as a non-privileged user, provided you first set your read privileges to m≥3. You do this with the following command:
`/MODIFY-TEST-OPTIONS PRIVILEGE=*PARAMETERS(READ=m,WRITE=1)`
You are only authorized to do this if you have been assigned this option for setting privileges in the user catalog.
If you have the right privileges (see above), you may convert a user dump into a system dump (with the MODIFY-TEST-OPTIONS command and the DUMP=*SYSTEM operand). The message `IDA0N45` is suppressed. The operator can control whether a system dump is stored on disk or tape.
In the event of an aborted system dump, message `IDA0N99` is output, depending on the system parameter DUMPCTRL.

A system dump that was stored on tape must be copied to a disk with the COPY-FILE command. It can then be edited with the DAMP diagnostic routine. The PERCON utility routine cannot be used to transfer the dump from tape to disk in this case.

Example

```
/IMPORT-FILE SUPPORT=*TAPE
           VOLUME=<volume>,DEVICE=<device>, FILE-NAME=<filename>  ------  (1)
/ADD-FILE-LINK LINK=DMCOPY11, FILE-NAME=<filename>,ACCESS-METHOD=*UPAM,
           BUF-LEN=*STD(2)  ---------------------------------------------  (2)
/ADD-FILE-LINK LINK=DMCOPY22, FILE-NAME=<output-filename>,
           ACCESS-METHOD=*UPAM,BUF-LEN=*STD(2)  ------------------------  (3)
/COPY-FILE FROM-FILE=<original-filename>,TO-FILE=<output-filename> -----  (4)
```

(1) Entry of the tape file in the system catalog.

(2) Specification of the link name DMCOPY11 for the tape file.

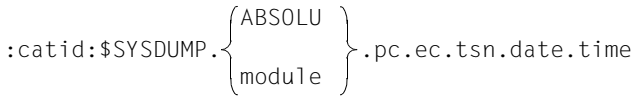
(3) Specification of the link name DMCOPY22 for the output file.

(4) The system dump is then copied from tape to disk using COPY-FILE.

See also the description of COPY-FILE in the “Commands” manual [8].

File name of a system dump

CDUMP saves a system dump under the SYSDUMP system ID and forms the file name of the system dump in accordance with the following basic pattern:



Where:

catid	is the catalog ID of the public volume set on which the dump is stored. If the dump was output to tape, no catalog ID is displayed.
module	is the name of the module from which the dump is activated, max 8 characters.
ABSOLU	is used if no name for the module exists.
pc	is the address in the program counter (relative to the start of the module or absolute).
ec	is the event code (hexadecimal). If the system dump was initiated with the CREATE-DUMP command, the event code has the value 'C8'.
tsn	is the TSN of the activating task.
date	is the date in the form Dyymmdd (D=marker identifying the start of a date, yy=year, mm=month, dd=day).
time	is the time in the form hhmmss (hh=hours, mm=minutes, ss=seconds).

4.2 Influence of page attributes

Memory pages can be provided with attributes. The following page attributes affect the processing of CDUMP:

– **Secret pages**
These are marked as such by the user by means of the CSTAT macro and then enjoy special protection.
Depending on what has been specified for the DUMPSEPA system parameter (see page 38), all secret pages, only secret pages from selected memory classes, or no secret pages will be included in the dump file.

– **Trusted pages**
These are provided with a memory protect key which is not the same as that of the user (e.g. pages used by the software product openUTM).
Pages of this type for a task are handled as follows:

If an **area or user dump** is called from a program section marked as “trusted”, the trusted pages are included in the dump and the dump is stored under the user ID SYSUSER.
If an area or user dump is called from a program section that has the user’s normal memory protect key, the trusted pages are excluded from the dump (exception: common readable pages, see below), where they are marked as “not accessible”. The area or user dump is then stored under the user ID.

All trusted pages are included in a **system dump**.

– **Common readable pages**
These can be marked as such by the owner by means of the CSTAT macro (class 6 memory), which makes them generally accessible. Common readable pages are included in the dump even if they are marked as trusted pages. However, if they are also marked as secret pages, they are excluded from the dump.

– **Read-protected areas**
If programs or parts of programs indicated as being subject to special protection (e.g. read-protected programs) are included in the dump, the user dump or area dump is not output to the user’s ID, but to the ID SYSDUMP instead.

For example, a program or part of a program deserves special protection if it is loaded or reloaded from a file that is protected with a read password and the user has not used the /ADD-PASSWORD command to specify the password.

– **Freshly-obtained pages**
  These are pages requested by and reserved for a user (allocated pages), but which have not yet been used.
  Freshly-obtained pages are not transferred directly to the dump file by CDUMP, but are only marked as “freshly-obtained” in the index structure for the person evaluating the dump.

– **DIV pages**
  DIV pages constitute a data area from a file which is mapped and processed in virtual memory.
  These pages can be included in the user dump if the user so wishes (DIV operand in the CDUMP2 macro with SCOPE=*USER). If the DIV operand is not specified in the macro call, the value set using the DATA-IN-VIRTUAL operand in the MODIFY-TEST-OPTIONS command determines how DIV pages are handled in a user dump.
  DIV pages are always included in a system dump.

– **MMAP pages**
  MMAP pages are data areas of POSIX files that are mapped and processed in the virtual address space. These pages can be included as a part of the user dump if requested by the user (MMAP operand in the CDUMP2 macro). If the MMAP operand is not specified in the macro call, the value set using the MEMORY-MAP operand in the MODIFY-TEST-OPTIONS command determines how MMAP pages are handled in a user dump.
  MMAP pages are always included in a system dump.

Note

User memory areas can also be located in data spaces. All the pages in a data space have the same page attributes with the exception of “allocated” and “freshly obtained”. (This does not apply to the pages in a particular program space.)

– Users can choose to include memory areas from data spaces in the **user dump** (DS operand in the CDUMP2 macro with SCOPE=*USER). If the DS operand is not specified when the macro is called, the value set using the DATA-SPACES operand in the MODIFY-TEST-OPTIONS command determines how data spaces are handled in a user dump.

– An **area dump** includes precisely those areas from data spaces specified by the user when the CDUMP2 call was issued (DSCTRL operand in the CDUMP2 macro with SCOPE=*AREA).

– A **system dump** only contains those areas from data spaces which the operating system passed to CDUMP via an interface ($DMPDEF(I)).

4.3 Controlling the CDUMP functions

The CDUMP functions can be controlled by means of system parameters and task-specific settings.

4.3.1 Control by means of system parameters

The system parameters are set in the startup parameter service and affect dump output throughout the system.

- A corresponding setting for the **DUMPSEPA** system parameter enables the output of protected memory areas (secret pages, see page 36) of the class 1 and class 6 memory to be suppressed.
- The output of privileged class 5 memory in area and user dumps can be controlled by setting the **DUMPCL5P** system parameter.
- The **DUMPSD#** system parameter defines how many system dumps per session are to be output without interaction from the operator.
- The following functions can be controlled using the **DUMPCTRL** system parameter:
  - detection of duplicates with system dumps (default: switched off)
  - unmanned operation (default: switched off)
  - output of the `IDAON99` message (default: switched off)
  - output of the `IDAON52` message (default: switched on)
  - use of the IOPERF= HIGH and IOUSAGE=WRITE operands in the FILE call for the dump output file (default: switched on)
- Output of class 6 memory in system dumps can be controlled using the **DUMPSREF** system parameter.
- The **RDTESTPR** system parameter defines the maximum read test privilege values.

  A read test privilege value of ≥ 3 is required in order to create a system dump from the processor state TU or to convert an area dump or user dump into a system dump.
- If the **DESTLEV** system parameter was set to ≥ **2**, the dump files are created under the SYSDUMP and SYSUSER user IDs with the operand DESTROY-BY-DELETE=YES.

More detailed information on the possible values and operands for system parameters can be found in the “Introduction to System Administration” manual [6].

4.3.2 Control by means of task-specific settings

Task-specific settings used when creating the dump can be specified with the command MODIFY-TEST-OPTIONS (see the “Commands” manual [8]).

The USERDUMP-OPTIONS operand controls if the output of the user and area dumps are for your own task or for other tasks running under your user ID (this restriction does not apply to user IDs with the TSOS privilege).
The user can specify:

– if any user or area dumps are to be made or if they are to be converted to system dumps (DUMP=*SYSTEM),

– on which pubset the dumps are to be stored

– if dump duplicates are to be suppressed or not,

– the maximum number of dumps that can be made and

– if the user dump is to contain DIV windows, data spaces or POSIX memory mapping areas.

The local task test privileges can be used with the PRIVILEGE operand to control if:

– user or area dumps are allowed to be converted to system dumps (DUMP=*SYSTEM) and

– if system dumps are allowed to be requested for the CDUMP2 macro or for the message `IDA0N45`.

A read test privilege ≥ 3 is necessary, either in the task in which the dump occurred or in the task that specified DUMP=*SYSTEM.

A requirement to be able to set the read test privilege ≥ 3 locally for a task is that the user ID under which the task is running has a sufficiently large value (see the ADD-USER or MODIFY-USER-ATTRIBUTES commands, TEST-OPTIONS operand).

4.4 Dump-specific operands in BS2000 commands

ADD-USER / MODIFY-USER-ATTRIBUTES command

System administration uses the ADD-USER command to define which test privilege a user is to be granted, see the manual “Commands” [8]:
`ADD-USER TEST-OPTIONS=*PARAMETERS(READ-PRIVILEGE=...,MODIFICATION=...).`

User-specific test privileges can be modified with the MODIFY-USER-ATTRIBUTES command.

You can display the current settings for privileges on a user-specific basis using the SHOW-USER- ATTRIBUTES command, TEST-OPTIONS operand (and on a task-specific basis using the SHOW-TEST-OPTIONS command, INFORMATION= *PRIVILEGE operand).

MODIFY-TEST-OPTIONS command

The user can employ the MODIFY-TEST-OPTIONS command to specify task-specific settings for test and diagnostics.

DUMP operand

The DUMP operand of the MODIFY-TEST-OPTIONS command is used to determine how the user dump generated with CDUMP is to be handled after the message `IDAON51 PROGRAM INTERRUPT AT LOCATION ...` is issued. It is also used to indicate whether a user or area dump is to be converted to a system dump.

You can display the current settings for outputting user and area dumps using the SHOW-TEST-OPTIONS command, INFORMATION=USERDUMP-OPTIONS operand.

*DUMP=*STD (interactive mode)*

CDUMP issues the following message:

```
IDAON45 DUMP DESIRED? REPLY (Y=USER-/AREADUMP TO DISK;
                      Y,<VOLUMETYPE>=USER-/AREADUMP TO TAPE;
                      Y,SYSTEM=SYSTEMDUMP TO DISK;
                      N=NO)
```

If the user responds with “N”, the dump is suppressed. If the user responds with “Y” or “Y,SYSTEM”, CDUMP generates the dump and issues the message:
`IDAON53 DUMP BEING PROCESSED. PLEASE HOLD ON`
If the user has a read test privilege ≥ 3 and responds with “Y,SYSTEM”, a user or area dump is converted into a system dump.
If `<VOLUMETYPE>` is specified, the dump is written to tape.

Message IDA0N53 is suppressed if CDUMP is called with the command CANCEL-JOB DUMP= (see the CANCEL-JOB command below).

*DUMP=*STD (in batch mode and in procedures)*

Output of the dump is suppressed and the following message is issued:
IDA0N48 TASK/SYSTEM SETTINGS PROHIBIT DUMP

*DUMP=*YES*

The user and area dumps are generated and the following message output to SYSOUT: IDA0N53 DUMP BEING PROCESSED. PLEASE HOLD ON
Message IDA0N45 is suppressed.

*DUMP=*NO*

Generation of user and area dumps is suppressed and the following message is output to SYSOUT instead of message IDA0N45:
IDA0N47 DUMP PROHIBITED BY /MODIFY-TEST-OPTIONS COMMAND

*DUMP=*SYSTEM*

A user or area dump is converted to a system dump if the user possesses a read test privilege ≥ 3.
The message IDA0N45 is suppressed.

DUMP-CONTENTS operand

The DUMP-CONTENTS operand of the MODIFY-TEST-OPTIONS command can be used to control inclusion of DIV windows, data spaces and POSIX MMAP areas in the user dump. The DUMP operand must not be set to *NO or *SYSTEM if this control is desired.

*DUMP-CONTENTS=*PARAMETERS(DATA-IN-VIRTUAL=*NO/*STD/*YES)*

If the operand value is set to *NO, no page of the DIV area is included in the user dump. The *STD and *YES operand values have the same meaning and cause the inclusion of DIV windows in the user dump.

*DUMP-CONTENTS=*PARAMETERS(DATA-SPACES=*STD/*YES/*NO)*

If the operand value is set to *NO, no page of the DS area is included in the user dump. The *STD and *YES operand values have the same meaning and cause the inclusion of data spaces in the user dump.

*DUMP-CONTENTS=*PARAMETERS(MEMORY-MAP=*STD/*YES/*NO)*

The value *STD or *YES causes POSIX pages to be included in the user dump; the value *NO means that no page of the MMAP area will be included in the user dump.

TSN operand

The TASK-ID operand can be used to specify the task for which the dump output settings are to be changed.

*TSN=*OWN*

The changes to the dump output settings apply to the user's own task.

TSN=<alphanum-name 1..4>/<c-string 1..4>

The changes to the dump output settings apply to the task with the specified TSN. Users who have not been assigned the TSOS privilege may only modify the settings of tasks running under their own user ID.

OUTPUT-PUBSET operand

The OUTPUT-PUBSET operand is used to specify the pubset to which user dumps or area dumps are saved.

*OUTPUT-PUBSET=*DEFAULT-PUBSET*

A user dump or area dump is saved to the default pubset defined for the user ID of the task causing the dump, provided the dump does not contain data deserving special protection. Otherwise, the dump is saved to the default pubset of the SYSUSER user ID.

OUTPUT-PUBSET=<cat-id 1..4>

A user dump or area dump is saved to the specified pubset.

If the dump cannot be saved to the specified pubset (e.g. because insufficient space is available), it is saved to the default pubset.

MAXIMUM-NUMBER operand

The MAXIMUM-NUMBER operand can be used to set a maximum value for the number of user and area dumps to be created.

*MAXIMUM-NUMBER=*UNLIMITED*

There is no upper limit on the number of user dumps and area dumps.

MAXIMUM-NUMBER=<integer 1..255>

The number of user dumps and area dumps is limited to the value specified here.

*SUPPRESS-DUPLICATES=*NO/*YES operand*

The SUPPRESS-DUPLICATES operand is used to specify whether a user dump or area dump should be suppressed if it is a duplicate of a dump that has already been generated. A dump is considered to be a duplicate if it occurs at the same location in the same program and has the same interrupt weight as another dump.

PRIVILEGE operand

The operand PRIVILEGE = *PAR(READ=...) is used to set read task specific read test privileges.
Users without privileges can only request a system dump if they have a read privilege ≥ 3 (see the ADD-USER command).

CANCEL-JOB command

The CANCEL-JOB command cancels a job. The DUMP operand determines whether a dump should be generated for the job being canceled or whether a user dump or area dump currently in progress should be aborted.

DUMP operand

The DUMP operand defines whether a dump is to be output for the job which is to be canceled.

*DUMP = *NO*

A dump is not requested. If, however, a dump is already in progress for the job being canceled, it is completed in full.

*DUMP = *STD*

If the operand DUMP=*YES or DUMP=*STD was set with the MODIFY-TEST-OPTIONS command, a dump is generated (even if *STD would imply a *NO, which is the case in batch or procedure mode). No dump occurs if DUMP=*NO is explicitly specified. If DUMP=*SYSTEM is specified a system dump is generated.

*DUMP = *CANCEL-RUNNING-DUMP*

If a user dump or area dump is already in progress for the job that is to be canceled, the dump is aborted immediately and the dump file is deleted.

MODIFY-SYSTEM-PARAMETERS command

The MODIFY-SYSTEM-PARAMETERS command can be used by the system administrator to set the CDUMP system parameters.

4.5 Execution messages

The operator is informed via the console as to why the system dump has been taken.

Depending on the CDUMP2 parameter DIAG the following messages may or may not be displayed (only if the system dump was requested by means of the CDUMP2 macro):

```
IDAON50   SYSTEMDUMP CALLED AT LOCATION '(&OO)'
```

```
insert = {<CDUMP-ad>.(module+nnnnn),EC=...[,ELSN=elsnr#]} {[,CODE=code]     }
         {<CDUMP-ad>.(ABSOLUT),EC=...[,ELSN=elsnr#]     } {[,INSERT=insert]}
```

The following message is always output:

```
IDAON51   PROGRAM INTERRUPT AT LOCATION '(insert)'
```

```
insert = {<error-ad>.(module+nnnnn),EC=...[,ELSN=elsnr#]} {[,CODE=code]     }
         {<error-ad>.(ABSOLUT),EC=...[,ELSN=elsnr#]     } {[,INSERT=insert]}
```

CDUMP-ad	Hexadecimal CDUMP call address
Error-ad	Hexadecimal error address
module	Module name
EC	Event code
ELSN	Number of Error Logging Sequence Block
CODE	Cause of dump
INSERT	String specifying the cause

The following message is then generally displayed to ask whether the operator wants to generate a system dump and, if so, to which output medium (disk or tape):

```
IDAON52   SYSTEMDUMP DESIRED? REPLY (EOT=DISK; VOLUMETYPE=TAPE; N=NO)
```

The operator may enter the following:

<tsn>.EOT	⟶ Output to disk
<tsn>.<device-identifier>	⟶ Output to disk
<tsn>.N	⟶ Suppress dump

The following entries, among others, can be made for <device-identifier> (see the DEVICE-TYPE operand of the CREATE-FILE command in the “Commands” manual [8]):

TA	Any tape device
TAPE	Any tape device
T-C4	tape device of volume type TAPE-C4

As an alternative, responding to the following message (that appears when a user/area dump is output to SYSOUT) leads to the selection of the output medium:

```
IDAON45   DUMP DESIRED?
          REPLY (Y=USER-/AREADUMP TO DISK;
          Y,<VOLUMETYPE>=USER-/AREADUMP TO TAPE;
          Y,SYSTEM=SYSTEMDUMP; N=NO)
```

The messages are repeated if an invalid tape device identifier is entered.

If a valid tape device identifier is entered, but the requested device is not available, message `IDAON58` is output on the console and repeated until a correct response is entered.

```
IDAON58   TAPE DEVICE TYPE NOT AVAILABLE?
          REPLY: (EOT=OUTPUT TO DISK; <VOLUMETYPE>=OUTPUT TO TAPE; N=NO)
```

Messages `IDAON52` and `IDAON58` are followed by messages from device management and from the data management system for controlling dump output (such as MOUNT messages, volume requests etc.).

If an error occurs in the system during dump output, error information (possibly together with an error code) is included in a message (`IDAON63`). Output is terminated if necessary (`IDAON61` message).

Correct termination of system dump output is indicated by the message:

```
IDAON54   SYSTEMDUMP WRITTEN TO FILE '$SYSDUMP....'
```

If the dump was output to tape, it must be copied to disk with COPY-FILE (and **not** with PERCON!) before it is processed by DAMP.

5 DAMP Dump analysis

Notes on the presentation of the version in this chapter:

- DAMP is used as the abbreviated form of **DAMP V4.9** (BS2000 OSD/BC V11.0).
- The character string <version> in sample outputs specifies the current version of DAMP, in this version of DAMP: <version>=V4.9A00.
- The character string <ver> in file names specifies the current version of DAMP, in this version of DAMP: <ver>=049.
- The character string <ver> in examples also specifies the current version of BS2000 OSD/BC (<ver>=20.0), of a dump generator, of VM2000, of system programs or of a diagnosis object.

5.1 Performance capabilities

DAMP is a program for analyzing diagnosis objects interactively. The diagnosis objects involved may be a dump file or an active BS2000 system.

DAMP can be used to virtually and symbolically analyze dump files containing a BS2000 system as well as an active BS2000 system. It also supports a real analysis of dump files which were created by some other operating system or which reflect a BS2000 system with damaged BS2000 structures. If the dump file structures are also corrupt, it may still be possible to perform an “emergency analysis” as a PAM file.

In order to work with DAMP, you will need to be familiar with the system and experienced in the field of diagnostics, since there is no provision for guided screen dialogs.

The program **DAMP** which is described here runs on the following servers and operating system versions:

- on /390 servers: BS2000/OSD-BC V9.0 and higher
- on x86 servers [1]: BS2000/OSD-BC V9.0 (OSD/XC V9.0) and higher

[1] On x86 servers, DAMP runs in emulated form

DAMP can process any dump files that were created on the following servers, regardless of which BS2000 version is currently active.

- on /390 servers as of BS2000/OSD-BC V8.0
- on x86 servers as of BS2000/OSD-BC V8.0 (OSD/XC V4.0)

DAMP requires the access method ANITA V20.0 to analyze dump files from BS2000 OSD/BC V11.0. ANITA V20.0 is available with a correction version for BS2000/OSD-BC V9.0 and higher.

5.1.1 Diagnostic log

All diagnostic activities and all screen outputs can be logged. All inputs and outputs are written into a file. This makes it possible to retrace the diagnostic course at a later point in time by analyzing the logged printouts or reviewing the logged session on the screen.

5.1.2 Creating lists

Since diagnosis is performed interactively on the screen, most of the lists needed for conventional offline diagnosis are superfluous. If such lists are nonetheless created, the type and scope of the information to be printed can be very easily specified.

5.1.3 Automating diagnostic processes

Actions which are repeated frequently can be automated. An automatic preliminary analysis can, for example, be used to identify the pages of a dump that are relevant for the diagnosis and to localize the error. By using the diagnostic language PRODAMP, the person performing the diagnosis can create procedures of instructions, including decision-based instructions, for diagnostic purposes. These procedures can then be saved in libraries and subsequently called as required.

5.1.4 Additional functions

If additional message and error files such as CONSLOG or SERSLOG are needed for the diagnosis, the editor EDT can be called as a subroutine. This editor can also be used to create or modify PRODAMP procedures.

DAMP can edit memory areas, for example in the layout of BS2000 dummy sections, by means of referring to elements in a symbol library.

In addition to the standard symbol library of the BS2000 basic configuration, users can also make use of their own private symbol files, generated by means of DAMP, for special analysis applications.

5.1.5 Behavior in the event of a program or system error

If an irregular program or system status is detected, DAMP outputs brief diagnostic information to SYSOUT and then terminates with a dump (user dump).

If, however, task switch 30 is set and DAMP is executing in interactive mode, a message is issued asking whether a dump should be generated.
If task switch 30 is set and DAMP is executing in batch mode, no dump is generated.

If you suspect an endless loop in DAMP, you should interrupt DAMP by pressing the function key K2. The rest of the run can be controlled with the INFORM-PROGRAM command (see the description of “Entering DAMP statements via the system command INFORM-PROGRAM” on page 224). Entering `/INFORM–PROGRAM MSG='?'` switches to SDF user guidance mode, and entering `/INFORM–PROGRAM MSG='*DUMP'` terminates the program with a dump.

5.1.6 Diagnosis objects that can be analyzed

5.1.6.1 Active system

The diagnosis object “active system” (CURRENT SYSTEM) always contains a BS2000 system. During the diagnosis, bear in mind that the system continues to operate even during the diagnostic session and is thus subject to constant changes. In order to diagnose the active system, you will need to have read test privilege 8 (see also the MODIFY-TEST-OPTIONS and SHOW-TEST-OPTIONS commands in the “Commands” manual [8]).

5.1.6.2 Dump files

DAMP can analyze not only dump files containing a BS2000 system, but basically all files that can be recognized by DAMP as a BS2000 dump file due to their metadata and analyzed as “real”. A BS2000 system can be analyzed virtually and symbolically.

SLED and SNAP

The diagnosis objects SLED and SNAP are created by dump file generators of the same names and can be analyzed as “real” or “virtual”. A SNAP always contains the BS2000 operating system. A SLED could also have been generated for other operating systems. Both types of dump files are created when the system is in an inactive state:

- SLED, following a system abort
- SNAP, on stopping the system for a brief period (up to 24 seconds)

The scope of both these dumps can be controlled via parameters, but the SNAP is subject to significant restrictions due to the time limit.

A SLED may contain multiple products:

- Complete VM2000 SLED

  A complete VM2000 SLED is a SLED file that contains the data of the VM2000 product as well as the data of the VM2000 guest systems. VM2000 guest systems run as virtual machines (VM) on a VM2000 system and may include both BS2000 as well as other operating systems.

- SLED from a SLED

  A SLED from a SLED is created if a further SLED was loaded before running the SLED. A SLED from a SLED is a file containing the data of the product SLED as well as the data of the product or products (for VM2000) that was run earlier.

System dump, user dump and area dump

The diagnosis objects SYSDUMP, USERDUMP and AREADUMP are created by the dump file generator CDUMP. They always contain data of the BS2000 and can only be virtually analyzed. The data is generated during the session and includes the memory areas of a task and selected system areas. The system dump provides diagnostic documentation for the system diagnostics staff; the user dump contains such details for the nonprivileged BS2000 user. An area dump contains memory areas defined by nonprivileged users.

SELF-LOADER

Any dump file that can be processed with DAMP as “real” can be opened by the user with the SELF-LOADER attribute. This enables the real-time analysis of dump files (SLEDs and SNAPs) that contain some other operating system or a BS2000 system with damaged BS2000 structures.

5.1.6.3 PAM file as diagnosis object

Even files that do not have the BS2000 dump format can be processed with DAMP (with some restrictions). In order to do this, they must be opened with the PAM attribute. This function is primarily intended to enable an “emergency analysis” of damaged dump files.

5.1.7 Online helps

During a diagnostic session you can request help from DAMP if you require this. If you enter a question mark (?) in one of the input fields and press the DUE key, DAMP supplies a description of this field. To do this, DAMP switches to EDT. You can scroll to find more detailed information on DAMP there. After EDT has terminated you can continue your diagnostic session. EDT mode '@VDT F2' is required for this help function. This is only available on 9763 data display terminals.

Online helps are available in German and English.

A further (less user-friendly) help option is provided by the help window (see page 60).

5.1.8 Terms used

DAMP screen	All data displayed on a screen by DAMP. The contents of one or more diagnostic windows can be displayed simultaneously on a DAMP screen.
Diagnostic window	The attributes of a diagnostic window are the name (W0 - W9, W21 - W99), the contents, and the representation of the contents. One section of the diagnostic window (W0 - W3) is allocated to fixed contents, the rest (the dump windows W4 - W9 und W21 - W99) can be allocated to different contents. Generally only a segment of the total contents of the diagnostic window is displayed on the DAMP screen.
Dump window	Dump windows are windows with the name W4 - W9 or W21 - W99. Depending on the contents of the window, they are also referred to as standard dump windows or special windows.

5.2 Screen format

5.2.1 Screen mask

The DAMP screen mask has a uniform structure and the lines described below always have the same meaning:

1 **Title line**
Displays the DAMP version and metadata on the diagnosis object.

2-3 **Message lines**
Output lines for DAMP and system messages.

4-22 **Diagnostic area**
Displays one or more diagnostic window(s), each with its own header line(s) and a dividing line between the windows.

23 **Command line**
Input line for DAMP statements and system commands.

24 **Key line**
Shows the content of each of the 9 diagnostic windows W1 - W9 which can be set using the P keys (the content of the other diagnostic windows is shown only in window W0).

```
line
1    DAMP <version> SYSDUMP(<ver>) from BS2000(<ver>)    <date> <time>
2
3
4    Analyzed Object: BS2000   V<ver>A00G1            TID=000A00CB TSN=HSMS W2,PLK,L
5    8
6    Dumpfile: :1DQM:$DIAGDUMP.A0550438.DHSIHSM@.00502.50.HSMS.D06081 on /390-HSI/VM
7    MemSize: 256.0 MB  ShareB: 00C00 UserXB: 01000 SysB: 71000
8
9    PCB#  PCB-Ad    IS-LNK    SR         Program Counter          SVC/IW      A_MODE
10      1  739073A8  72F91008  070C0C00   71542CBE=CDUMPF1 +0117E  EE=$PNUP        31
11      2  72F91008  732DC578  070C0C00   7139ECB0=NRTINIT +01670  1A=CDUMP        31
12      3  732DC578  00000000  070C0C00   7C584BC2=DHSIHSM@+00502  ED=TPR-Ter      31
13   ------------------------------------------------------------------------------
14   ETCB                            +00344=720E2AEC TID=000A00CB          W5,CBA,L 3
15   344 ETCBSTA : 739073A8                 | 348 ETCBTRC@: 00000000 =     0
16   34C ETCBCTRC: 71FEC420                 | 350 ETCBADM1: 00000000 =     0
17   ------------------------------------------------------------------------------
18   DHSIHSM@                        +004FC=7C584BBC TID=000A00CB          W4,ASS,L 6
19   7C584BBC (04FC):   41 10 DD88        =   LA    R1,3464(,R13)
20   7C584BC0 (0500):   0A ED             =   SVC   237                   (TPR-Ter)
21   7C584BC2 (0502):   98 A6 D0D0        =   LM    R10,R6,208(R13)
22   7C584BC6 (0506):   07 FE             =   BR    R14
23   7C584BC8 (0508):   00060202          =   DC    X'00060202'
24   CMD:
     Key: 1=Help 2=Plk 3=PCB 4=U7C584 5=ETCB   6=EXVT   7=MEMA   8=SUSY   9=FIND
```

Figure 1: The DAMP screen mask

The mask in figure 1 contains the following elements:
title line (line 1), two message lines (2-3), status window W2 with a length of 8 lines (4-11), dividing line (12), window W5 in symbolic format with a length of 3 lines (13-15), dividing line (16), window W4 in assembler format with a length of 6 lines (17-22), command line (23) and key line (24).

The various lines in the DAMP screen mask have the following functions:

The title line (line 1)

The title line displays:
– the DAMP version
– the type of the analysis object
– the dump generator version (only for a dump file)
– the name and version of the product contained in the diagnosis object
– the date and time the diagnosis object was created

The second field (type of the analysis object) can contain the following information:
– CURRENT SYSTEM
– SLED (including complete VM2000 SLED and SLED from a SLED)
– SNAP
– SYSDUMP
– USERDUMP
– AREADUMP
– SELFLOADER
– PAM FILE

See also section “Diagnosis objects that can be analyzed” on page 50.

If a diagnosis object (e.g. a VM2000 SLED file) includes data from multiple products, the name and version refer to the product that that first appears on opening the object (in the case of a VM2000 file, this is the product VM2000). This information remains in the title line even if some other product from the object is subsequently selected.

Further information relating to the analysis object can be output in INF mode in the status window (W2).

```
DAMP <version> SYSDUMP(<ver>) from BS2000(<ver>)    <date> <time>
```

Figure 2: Title line when a system dump is the diagnosis object

```
DAMP <version> CURRENT SYSTEM from BS2000(<ver>) <date> <time>
```

Figure 3: Title line when the active system is the diagnosis object

The message lines (lines 2 and 3)

The message lines display messages from the DAMP system. You can use program keys P14 and P15 to scroll backward or forward in the message history.

The user option “Blinking” can be used to activate and deactivate flashing messages (see “Column separator (list)” on page 135).

```
DAMP <version> SYSDUMP(<ver>) from BS2000(<ver>)    <date> <time>
DMP8751 CLASS  1 SEARCH INTERRUPTED; <F3> CONTINUE, <F1> CANCEL

FIND - Command                                  SYS=000A00CB          21,D  ,L16
```

Figure 4: Messages from the DAMP system in lines 2 and 3

In EDT mode, these messages are output on the last two data lines of the EDT screen.

The diagnostic area (lines 4 to 22)

The requested diagnostic windows (see page 58) are displayed in the diagnostic area of the mask. These contain either the information from the dump under examination or various help texts. This dividing line can be optionally deactivated. The substitution characters for nonprinting characters and for window and column separators can be set according to the generated terminal type.

The command line (line 23)

The command line starts with **CMD:** and is used for the entry of DAMP statements and the permitted system commands (see page 167 for a list of possible DAMP statements).

```
71A6C336 (016E): D12C0DEF  D5014006  A7444780  A0C6D203   <==>  J...N. .x....FK.
71A6C346 (017E): D13CA50C  47F0A0CC  D203D13C  400C4140   <==>  J.v..0..K.J. ..
CMD:
Key: 1=Help 2=Plk 3=PCB 4=Dump   5=ETCB   6=Dump   7=Dump   8=U71A6C 9=Dump
```

Figure 5: Input area for DAMP statements and system commands

If a system command entered in the DAMP mask results in a system message, this message is displayed in lines 2 and 3 (the message lines).

If you enter a question mark or the name of a statement followed by a question mark, the system switches to SDF user guidance mode. This mode displays screen masks which allow you to complete the statement by filling in the fields required and to then pass the completed statement to DAMP by means of the execute function.

BS2000 system commands can be entered directly in the command line. If ambiguities arise because a command has the same name as a DAMP statement, the command can be prefixed by a command label such as /LABEL, which ensures that the command is then interpreted as a system command.

The key line (line 24)

The key line shows the content of each of the 9 diagnostic windows W1 to W9 which can be set using the P keys, regardless of whether or not these windows are visible on the screen. The assignment of the other diagnostic windows is shown only in window W0. The description of the diagnostic windows is provided on page 58.

To start with, the dump windows (W4 - W9 and W21 - W99) are freely available standard dump windows. This can be seen for the dump windows W4 - W9 by means of the “Dump” indicator in the key line.

If one of the dump windows (W4 - W9) was being used and this window is not a special window, the page of the system memory or user memory of which a part is currently displayed in this window is displayed in the corresponding position in the key line.

The key line contains the following display:

Snnnnn	for system memory
Unnnnn	for user memory
Dnnnnn	for data spaces

If this part of the page is displayed in symbolic format, the name of the control block name is shown (abbreviated, if necessary, to 6 characters).

If hardware information is being displayed, the appropriate field in the key line shows one of the following:

Rnnnnn	for output with real addressing (ASEL=RM); nnnnn: address specified relative to the associated 4GB segment
Annnnn	for output with absolute addressing (ASEL=ABS); nnnnn: address specified relative to the associated 4GB segment
Hnnnnn	for output of the hardware system area (ASEL=HSA)
PSSnnn	for output of the processor saved status (ASEL=PSS) nnn: processor number
snnnnn	for output of a dump file section (ASEL=SCT)

If a special function is assigned to a window, this function is also shown in the appropriate position in the key line, namely:

AUDI	for	SHOW-EDITED-INFORMATION INFORMATION=*AUDIT-TABLE-EDIT
FILE	for	SHOW-EDITED-INFORMATION INFORMATION=*DUMPED-SYSTEM-FILE
FIND	for	START-PATTERN-SEARCH
LIST	for	START-LIST-GENERATION
MEMA	for	SHOW-EDITED-INFORMATION INFORMATION=*MEMORY-ATTRIBUTES
OPTS	for	START-OPTION-DIALOG
PROC	for	START-PRODAMP-EDITOR
SUSY	for	SHOW-EDITED-INFORMATION INFORMATION=*SUBSYSTEM-INFORMATION
TABL	for	SHOW-EDITED-INFORMATION INFORMATION=*TASK-TABLES
TRAC	for	SHOW-EDITED-INFORMATION INFORMATION=*TRACE-TABLE-EDIT

It is thus possible to see, at any time, which diagnostic window contains which memory segment and which diagnostic windows are still available for assignment. The statement `SHOW-EDITED-INFORMATION INFORMATION=*STORAGE-EDIT` can be used to cancel use as a special window, in which case “Dump” is displayed again.

```
068 ETCB@19@ : 70F8E000                | 06C ETCB@20@ : 70F81F40
070 ETCB@21@ : 00000000 =     0        | 074 ETCB@22@ : 00000000 =     0
CMD:
Key: 1=Help 2=Plk 3=PCB 4=S71234 5=D705FE 6=U70F12 7=ETCB   8=Dump   9=TRAC
```

Figure 6: Key line

Line 24 is the key line and shows the following:

– windows 1 to 3 have fixed assignments, namely as help, status and stack windows
– window 4 contains virtual system page 71234
– window 5 contains virtual system page 705FE of a data space
– window 6 contains user page 70F12
– window 7 contains the TCB of the current task in symbolic notation
– window 8 is not used
– window 9 is occupied by the trace table

5.2.2 Diagnostic windows

A total of 89 diagnostic windows are available for diagnosis on the screen. The windows are designated W0 through W9 and W21 through W99 (W10 through W20 cannot be used as diagnostic windows). These windows consist of

- the overview window (W0)
- the help window (W1)
- the status window (W2)
- the stack window (W3)
- the dump windows (W4 - W9 and W21 -W99)

A maximum of 19 lines of one or more windows, including the dividing and header line(s), can be displayed on the screen.

The windows can be assigned contents during the diagnosis session:
Window W0 contains the assignment of all 89 diagnostic windows, window W1 is assigned permanently to the online help function provided by DAMP, windows W2 and W3 are used when a BS2000 diagnosis object is opened, while windows W4 to W9 and W21 to W99 can be assigned freely, e.g. with the `SHOW-EDITED-INFORMATION` or `START-PATTERN-SEARCH` statement.

You can control the number, order and length of the windows displayed on the screen with the `MODIFY-SCREEN-LAYOUT` statement. Any assignment made for a window remains valid even if the window is currently not visible. As an alternative to the `MODIFY-SCREEN-LAYOUT` statement a window (of the currently valid length) can also be made visible by pressing a P key (windows W1 through W9) or by entering the window number (0 - 9, 21 - 99) in the command line and pressing DUE.

For further information on the diagnostic window, see also page 84.

5.2.2.1 The overview window (W0)

The overview window (W0) always contains the current assignment of the available diagnostic windows W0 through W9 and W21 through W99. As W10 through W20 may not be used as diagnostic windows (to prevent conflict situations with the current use of P keys P10 through P20), these must be marked as “reserved”.

If a diagnostic window is currently not assigned, this is displayed in window W0 by the output of blanks, otherwise the current window assignment is displayed. The display takes place in the key line (see the description of the “The key line (line 24)” on page 56) analogously to the display for windows W1 through W9. Since more characters are available for the assignment display in window W0 than in the key line, the address (instead of the page number) is displayed here for memory areas.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


DAMP Window Assignment                                                 W0,WIN,L19
00 = WINDOWS    01 = HELP       02 = PLK        03 = PCB        04 = U70FFD328
05 = S728D73E8  06 =            07 = U70FFCFC8  08 = D00000000  09 =
10-20: reserved 21 = EXVT       22 = ETCB       23 = S728BA380  24 =
25 =            26 =            27 =            28 =            29 =
30 =            31 =            32 =            33 =            34 =
35 =            36 =            37 =            38 =            39 =
40 =            41 =            42 =            43 =            44 =
45 =            46 =            47 =            48 =            49 =
50 =            51 =            52 =            53 =            54 =
55 =            56 =            57 =            58 =            59 =
60 =            61 =            62 =            63 =            64 =
65 =            66 =            67 =            68 =            69 =
70 =            71 =            72 =            73 =            74 =
75 =            76 =            77 =            78 =            79 =
80 =            81 =            82 =            83 =            84 =
85 =            86 =            87 =            88 =            89 =
90 =            91 =            92 =            93 =            94 =
95 =            96 =            97 =            98 = TRACCMD    99 = FINDCMD
CMD:
Key: 1=Help 2=Plk 3=PCB 4=U70FFD 5=S728D7 6=Dump   7=U70FFC 8=D-0000 9=Dump
```

Figure 7: Output in the overview window (W0)

To select a window for output on the screen, mark a window number with MAR and then press DUE. Alternatively a window can also be selected in the manner described under “Diagnostic windows” (see page 58).

5.2.2.2 The help window (W1)

The help window is used to display information on the functions and operation of DAMP. This information is divided into chapters and sections, with the chapter currently on display being shown in the header line (“Chapter nnnn”).
The help information is available in German and English.

Keywords in the help information are displayed with high intensity and can be marked. rking such a keyword with the MAR key and pressing DUE causes the chapter or section containing more detailed information on this keyword to be displayed.

The following paging functions are available to you in the help window:
– to page to the start or end of a chapter, enter “--” or “++”
– to page n lines back or forward, enter “-n” or “+n”
– to page back or forward by one window length, enter “-” / F1 or “+” / F3

```
DAMP <version>  No Object opened in BS2000 V<ver>  <date> <time>


H E L P    Kapitel 0001 DEUTSCH ----------------------------------     W1,TXT,L19
                        D A M P   Version <version>
                 Dump Analysis and Maintenance Program.

Dieses Programm dient zur Auswertung von Diagnoseobjekten (Dumpdateien und
aktives System) im Dialog. Die folgenden Dumpdateitypen werden unterstuetzt:
SLED (auch VM2000-Gesamtsled), SNAP, Systemdump, Userdump und Areadump.
Zur Diagnose des aktiven Systems ist die Lese-Testprivilegierung 8 erforderlich.

  Weitere Informationen: Inhalt / Stichwoerter markieren oder Kapitel angeben,
                         Blaettern mit --, ++, - (F1), + (F3), -n, +n
  Beginn der Auswertung: Anweisung oder '?' in CMD-Zeile eingeben,
                         Diagnosefenster mit P-Taste oder Fenster-Nr. auswaehlen

Durch Eingabe eines '?' in ein Eingabefeld eines Diagnosefensters erhalten Sie
direkte Hilfe zu diesem Feld (nur bei DSS 9763).

Note for English users:
By entering 'ENGLISH' in the header line you can change the language!

CMD:
Key: 1=Help 2=Plk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=Dump   9=Dump
```

Figure 8: Output in the help window (W1)

The help window has the input fields “Chapter,” “Language” and “Window length”, which are located in the header line (see next page):
– You can select each chapter directly by entering the chapter number in the “Chapter” input field.
– You can switch the language of the display in the help window, of the online help texts (which are displayed in an input field when you enter “?”) and of the DAMP messages from GERMAN or ENGLISH (default) in the “Language” input field. You can also change the language using the special window OPTIONS (see page 133).
– The length of the help window is entered or displayed in the “Window length” field.

Figure 9: Input fields in the header line for the help window

In addition to the use of the help window, DAMP also offers the - often more user-friendly - option of calling help information by entering a question mark ('?') in an input field of one of the diagnostic windows (see page 52). However, the '?' help is only available on 9763 data display terminals (with EDT mode '@VDT F2') in German and English.

5.2.2.3 The status window (W2)

The status window provides an overview of the opened diagnosis object. The first four lines contain general information on the type of diagnosis object and on its environment during creation (e.g. BS2000 version, CPU type or memory size).

The status window is displayed automatically on opening the diagnosis object and contains the input fields “TID”, “TSN”, “Mode select” and “Window length”.

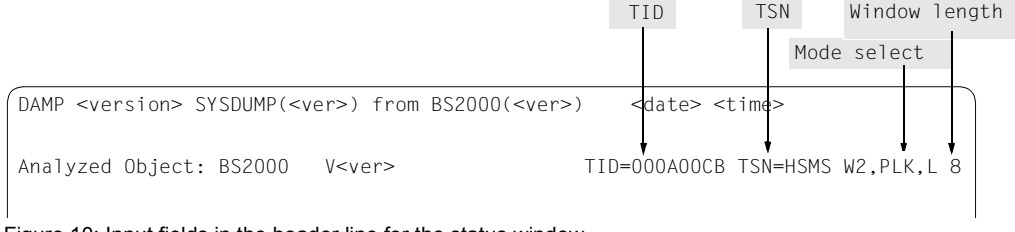


Figure 10: Input fields in the header line for the status window

Mode select

The information shown in the “Mode select” input field for the status window (W2) can be influenced by different modes. The possible modes are INF, TSK, PLK and SLK:

INF Besides the standard details displayed in the first three lines of the status window, additional information on the dump generator, diagnosis object and selected product is returned. The INF mode is set automatically if the dump file contains more than one object (product), e.g. in the case of a complete VM2000 SLED.

DAMP can analyze the following objects:
– active BS2000 system
– system, user and area dumps with a BS2000 dump object
– SNAP dump with a BS2000 dump object
– SLED with a BS2000 dump object
– SLED with a VM2000 dump object (complete VM2000 SLED)
– SLED with a SLED dump object (SLED from a SLED / Dump from a SLED)
– SLED with some other dump object such as SIR, for example
– virtual machine (VM) under VM2000
– predecessor system in SLED
(predecessor systems are BS2000, VM2000 or other systems)

Example windows for the mentioned object types can be found as of page 63.

The mode can be set by the user by means of the “INF” entry in the header line. If an object selection was made in dump files with multiple objects (complete VM2000 SLED, SLED from a SLED / Dump from a SLED), entering “-” / F1 in INF mode cancels the selection.

In INF mode, as much information as possible is displayed. One exception is the so-called SELF-LOADER (see page 141).
The following information is currently displayed:
– the name of the dump file and the HSI of the analyzed object
– the memory sizes of the analyzed object
– the CPU type followed by `Virtual Machine` when a virtual machine is concerned
(after live migration, the new system name followed by `(after Live-Mig)` is also output here)
– the type and version of the dump generator of the object to be analyzed
– the contents of the product ID; included in this is the name and the version of the product, and if available, the address of the so-called dump testament (contains internal SLED information).

By marking (see section “Marking” on page 86) the address of the dump testament, the memory contents of the dump testament can be output to a standard window with the RM or ABS addressing mode.

Depending on the dump file type, the following information is included:

- For system, user, and area dumps
  the complete contents of the “dump title” dummy section.
- In the case of a SLED
  – The contents of the BS2000 crash message with printable text.
    A VM2000 crash message is not recognized by DAMP.
  – The contents of the Time of Day register is output in edited form.
  – Information relating to additional dump objects, for example, PREVIOUS SYSTEM or DUMPED SYSTEM. In the case of a VM2000 dump object, an overall view of the virtual machines in the overall system is offered. The required dump object is selected by means of marking.
  – If a STARTUP dump exists, a message.
- In the case of a SNAP dump
  – The address of the SNAP information.
  – The contents of the SNAP message.
  – Information relating to the $SNAP call, for example, the function area from which the call was started, the TSN of the caller, the address of the SNAP call, and the start address of the GP register record.
- If the file to be analyzed is opened as a PAM file, details on the opened file, e.g. the file name, the file size and the last-page pointer, are automatically provided in this mode.

Example windows for the various dump file types in INF mode

```
DAMP <version> SYSDUMP(<ver>) from BS2000(<ver>)    <date> <time>


Analyzed Object: BS2000     <ver>              TID=00010001 TSN=     W2,INF,L 8
Dumpfile: :1DQM:$DIAGDUMP.QM122.05.LOI.SLED.ABGQN406-8 on X86-HSI/VM
MemSize:   3.8 GB  ShareB: 00D00 UserXB: 01000 SysB: BF000
CPU: X86SU-  300-160F / Virtual Machine X86SU-  300-160F (after Live-Mig)
Generator Name: SLED     (STD)          Generator Version: <ver>
Product Name:   BS2000                  Product Version:   <ver>

Dumptitle:   TSN-HSMS   ELSN-          SYSTEMDUMP   PC- 7C584BC2(DHSIHSM@+00502
             )   EC-50   VERS-<ver>   DUMP-TIME   <time>  <date>




--------------------------------------------------------------------------------
```

Figure 11: Information screen in the status window (W2). Dump created by CDUMP.
Dump object: BS2000 (SU x86 after Live Migration)

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date>  <time>


Analyzed Object: BS2000   <ver>                TID=00010001 TSN=     W2,INF,L13
Dumpfile: :1DQM:$DIAGDUMP.A0610398.SLED.ABGSE21B.170208 on /390-HSI/VM
MemSize: 512.0 MB  ShareB: 00C00 UserXB: 01000 SysB: 71000
CPU: 390SU-  700-70 / Virtual Machine
Generator Name: SLED     (ALL )         Generator Version: <ver>
Product Name:   BS2000                  Product Version:   <ver>

Time of creating SLED:  <date>  <time>

ID of Crash Message: NRTT501 SETS.      Crash ID: NRTC515
Crash Insert: SVC ERROR AT NIA F1A67C44

Crash Caller: F1A67C44 = ETMOSVC  + 00204

--------------------------------------------------------------------------------
```

Figure 12: Information screen in the status window (W2). Dump created by SNAP. Dump object: BS2000 (SU /390 after abnormal system termination)

```
DAMP <version> SNAP(<ver>) from BS2000(<ver>)       <date> <time>


Analyzed Object: BS2000   V<ver>            TID=00010001 TSN=     W2,INF,L13
Dumpfile: :2OS6:$SPM0.MEN.SNAP.WILLI on /390-HSI/VM
MemSize: 256.0 MB  ShareB: 00C00 UserXB: 01000 SysB: 71000
CPU: 7.500- S210-40 / Virtual Machine
Generator Name: SNAP                  Generator Version: <ver>
Product Name:   BS2000                 Product Version:   V<ver>

SNAPID: NSPTEST
SNAP Insert:

Return Code of SNAP call: SNAP successfully processed
SNAP from SIH                         Address of SNAP call: 00000040
Address of SNAP internal data: 729A2800
--------------------------------------------------------------------------------
```

Figure 13: Information screen in the status window (W2). Dump created by SLED. Dump object: BS2000 (S server)

```
DAMP <version> SLED(<ver>) from VM2000(11.5)


Analyzed Object: VM2000   V<ver>            TID=         TSN=     W2,INF,L19
Dumpfile: :1DQM:$DIAGDUMP.QM113.36.SLED
                                                          (No Selection)

Generator Name: SLED     (ALL )         Generator Version: <ver>
Product Name:   VM2000                  Product Version:   V<ver>
Address of Dump-Testament: 00001024 (absolut)

Time of creating SLED:  <date>  <time>
Information about VM2000:    Hypervisor pages FROM 0000 TO 0DFF
VMs created by VM2000:   HYP  VM01 VM02 VM03 VM04


VMs dumped by SLED and their page boundaries
VM01: 000E00 - 00ADFF    VM02: 00AE00 - 014DFF    VM03: 014E00 - 037DFF
VM04: 037E00 - 0E6DFF



CMD:
Key: 1=Help 2=Inf 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=Dump   9=Dump
```

Figure 14: Information screen in the status window (W2). Dump created by SLED; dump object: VM2000

```
DAMP <version> SLED(<ver>) from SLED(<ver>)


Analyzed Object: SLED     V<ver>            TID=         TSN=     W2,INF,L11
Dumpfile: :2OS6:$SPM0.SLEDFROMSLED

CPU:
Generator Name: SLED     (ALL )         Generator Version: <ver>
Product Name:   SLED                    Product Version:   V<ver>
Address of Dump-Testament: 021D18C0 (real)

Time of creating SLED:  <date>  <time>

Choose diagnosis system:      DUMPED SYSTEM          PREVIOUS SYSTEM
--------------------------------------------------------------------------------
```

Figure 15: Information screen in the status window (W2). Dump created by SLED; dump object: SLED

```
DAMP <version> SLED(<ver>) from VM2000(<ver>)       <date> <time>


Analyzed Object: BS2000   V<ver>              TID=00010001 TSN=     W2,INF,L 8
Dumpfile: :1DQM:$DIAGDUMP.QM113.36.SLED on /390-HSI/VM
MemSize: 160.0 MB  ShareB: 00C00 UserXB: 01000 SysB: 71000   VM01
CPU: 7.500- S210-60 / Virtual Machine
Generator Name: SLED     (ALL )         Generator Version: <ver>
Product Name:   BS2000                  Product Version:   V<ver>

Time of creating SLED:  <date>  <time>
--------------------------------------------------------------------------------
```

Figure 16: Information screen in the status window (W2). Dump created by SLED; dump object: BS2000 in the monitor VM

```
DAMP <version> SLED(<ver>) from SLED(<ver>)         <date> <time>


Analyzed Object: BS2000   V<ver>              TID=00010001 TSN=     W2,INF,L 8
Dumpfile: :2OS6:$SPM0.SLEDFROMSLED
MemSize: 456.0 MB  ShareB: 00C00 UserXB: 01000 SysB: B0000   PREV SYS
CPU: 7.500- S210-60
Generator Name: SLED     (ALL )         Generator Version: <ver>
Product Name:   BS2000                  Product Version:   V<ver>

Time of creating SLED:  <date>  <time>
--------------------------------------------------------------------------------
```

Figure 17: Information screen in the status window (W2). Predecessor system; dump object: BS2000

TSK This mode is set automatically if a SLED or SNAP with a BS2000 object or the currently active system is to be analyzed and information relating to a number of tasks is available. Only the first 14 tasks are initially shown in the status window (W2). Each line contains the information for one task and can be marked.
The mode is indicated by the entry “TSK” in the key line.

You can scroll in the task list by entering +, -, ++, --, +n, -n in the command line and pressing DUE . The F3 or F1 key can be used instead of “+” DUE or “-” DUE.

i When diagnosing the active system, the task list is updated only on positioning with “--” to the beginning of the list (i.e. when TID 0001 is displayed).

When diagnosing the active system, the termination or creation of tasks can also result in inconsistencies in other windows. If necessary, these have to be updated by bringing the task list up to date.

In the event of a system dump the output of the task list can be set by entering “TSK” in the field mode selection (PLK mode is set by default for the 'error task'). For a system dump the task list also contains all tasks from the BS2000 system to be diagnosed. A task can be selected for further diagnosis by marking a line.

CAUTION!
The system dump contains system-wide data for all tasks, but only the local task data of the 'error task'. When the system data (e.g. for PCB chaining) of the other tasks is analyzed extra special care is called for as these tasks cannot be stopped when the system dump is taken.

The task list can be sorted according to various criteria. To do this you must select a column in the header line. Sorting takes place in ascending order in accordance with the content of the column selected. The default setting is sorting according to TID (1st column).

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>
DMP2307 TID 00010001 HAS BEEN SELECTED AS CURRENT TID

Analyzed Object: BS2000   V<ver>              TID=         TSN=     W2,TSK,L19
Dumpfile: :SLED:$DUMPFILE.SLED.CS503K on /390-HSI/VM
MemSize: 512.0 MB  ShareB: 00C00 UserXB: 01000 SysB: 71000

     TID  TSN Typ Q-PND Act SVC/IW   Current PCR     Caller SVC/PCR
00010001  TSC PA  12 17 FA=$BOWT     JSM@@@@@+00316
00010002 CLOG SYS 12 17 FA=$BOWT     NBRCLOG +003D0
00010003  RMM PA  12 17 FA=$BOWT     EMMRMRCM+02018
00010004 HERS SYS 11  6 EB=$PEND     EHERSPT +001BC
00010005  PT5 PA  13  4 59=VPASS     ETMPT5  +0027C
00010006  PT6 PA  12 17 FA=$BOWT     JSSTASK@+00952
00010007  PGE PA   4 17 FA=$BOWT     DJPGER  +010FC
00010008  UCO PA   4 17 FA=$BOWT     NBRMAIN +00048
00010009  REK PA  12 17 FA=$BOWT     ETMRK2B +00190
0001000A  VMM PA   0  0 F1=$PNDT     NRTSEH  +01662  48=Pag. Err  MESCPUSS+2593C
0001000B  MSG PA  12 17 FA=$BOWT     ETMRK2F +000EA
0001000C  KTT PA   4  4 59=VPASS     NBCADM  +01C5C
0001000D RUNT SYS 12 17 FA=$BOWT     ECCLP   +007D2
0001000E SEST SYS 12 17 FA=$BOWT     NBESSWR@+036FE
CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=Dump   9=Dump
```

Figure 18: Task overview in the status window (W2)

PLK This mode is set automatically if an area, user or system dump containing only one task is being edited. It can also be set by marking a task line or by entering a TID/TSN in the TSK mode. The right-hand part of the 4-byte TID is sufficient when selecting a task.
All PCBs associated with this task and the related information is then output. The edited output of a PCB can be displayed in the stack window (W3) by marking a PCB line.
The symbol `PLK` appears in the key line.

User PCBs are marked in the output line with an “*” (asterisk).

The column headed A_MODE in the PCB overview for x86 objects lists the context type as well as the addressing mode. CIS stands for a PCB with a /390 context and X86 for a PCB with an x86 context.. With these objects, the following output is possible:

– x86 systems' objects:
  CIS 31 or CIS 24 if the PCB has a /390 context (code executing in /390 mode – i.e. as an emulation – with 31-bit or 24-bit addressing)

– x86 systems' objects:
  X86 32, X86 31 or X86 24 if the PCB has an x86 context (code executing in x86 mode – i.e. natively – with 32-bit, 31-bit or 24-bit addressing)

You can return from the first PCB to the task overview with F1 / "-" or by entering TSK in the Mode select field.

```
DAMP <version> SYSDUMP(<ver>) from BS2000(<ver>)    <date> <time>


Analyzed Object: BS2000   V<ver>               TID=00010065 TSN=2WG7 W2,PLK,L 9
Dumpfile: :LOU3:$SYSDUMP.EVENT#SA.000D4.C8.2WG7 on X86-HSI
MemSize:   5.5 GB  ShareB: 00C00 UserXB: 01000 SysB: BF000
Job: MENCHER /TSOS    /ADMINSTR Cmd: CREATE-D Prg: SYSPRG.DAMP.<ver>
PCB#  PCB-Ad    IS-LNK    SR         Program Counter          SVC/IW      A_MODE
   1  C33271E0  C3327768  070C0C04   C0BE9094=CDUMPF1 +02B14  EE=$PNUP    X86 32
   2  C3327768  C3327590  070C0C04   C1D687E6=NSCDUMP +00206  1A=CDUMP    X86 32
*  3  C3327590  00000000  07ED2C00   010072D4                 5C=BKPT     CIS 31
   4  C33273B8  00000000  070C0C04   C05A456E=ETMSF   +008EE  E9=$FNAT    X86 32
--------------------------------------------------------------------------------
```

Figure 19: PCB overview in the status window (W2), dump file with X86 object

SLK In this mode, the call chain is output via the TPR program manager (SPL linkage). To do this, the mode field in the title line of the window must be overwritten with the symbol SLK. The edited output of this program manager stack can be displayed in the stack window (W3) by marking one of the stack lines shown.
The symbol SLK appears in the key line.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


Analyzed Object: BS2000   V<ver>               TID=000402FB TSN=0VGY W2,SLK,L19
Dumpfile: :SLED:$DUMPFILE.SLED.CS507K.1015 on /390-HSI
MemSize:   4.0 GB  ShareB: 00C00 UserXB: 01000 SysB: 71000
Job: NKI10882/DMS10   /A        Cmd: CALL-PRO Prg:
Stk#  Stack-Ad  ADF       Params (R1) Caller (R14)            ADF-Ind
   1  6F3D22D0  6F3D2328  6F3D2398    F14477FA=ECTYP   +037FA N     current
   2  6F3D2428  6F3D2480  6F3D23D0    F144587A=ECTYP   +0187A N
   3  6F3D2580  6F3D25D8  6F3D2FE4    F1444D82=ECTYP   +00D82 N
   4  6F3D2740  6F3D2798  6F3D2FE4    FF71922C=NMHRPUT +00BAC N
   5  6F3D3160  6F3D31B8  7F719F8A    FF71CD22=NMHRSPL +00A22 N
   6  6F3D3AB0  6F3D3B08  6F3D3BE0    F154BEFC=CDUMPF3 +02D7C N
   7  6F3D3C58  6F3D3CB0  00000002    F15453A4=CDUMPF1A+00B24 N
   8  6F3D3D80  6F3D3DD8  70FA2000    F1541E3E=CDUMPF1 +002FE N
   9  6F3FC2D8  6F3FC330  00000000    00000000                N
  10  6F3FD590  6F3FD5E8  6F3FEFE8    FD586B74=CLIKREA@+03E34 N
  11  6F3FF238  6F3FF290  6F3FFF68    FD56BBEA=CLIISL@@+007FA N
  12  70FDA870  70FDA8C8  70FDB1DC    F1390018=NLKISLS +00558 O
  13  70FDA938  70FDA990  70FDB1DC    FF23F57A=SSMLIBR@+0056A N
  14  70FDBE60  70FDBEB8  70FDD1B8    FF24BF8E=JSYLIBR +0004E N
CMD:
Key: 1=Help 2=Slk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=Dump   9=Dump
```

Figure 20: Chain of program manager calls in the status window (W2)

You can return from the first stack to the task overview with F1 / "-" or by entering TSK in the "Mode select" field.

Paging in the status window

You may enter the following: +, -, ++, --, +n, -n or press the function keys F3 (page forward by one window length, which corresponds to +) and F1 (page backward by one window length, which corresponds to -). For further details, see the section on "Paging in a diagnostic window" on page 86.

Note on paging forward

When paging forward (+, ++, +n, F3), you stop at the end of the listing involved, and you are not automatically returned to the start of the listing.

Note on paging backward

- Pressing the F1 key and entering “-” in the PLK and SLK modes pages back to the task list if the first PCB or first stack is currently visible in the window. Selecting the TSK mode also returns you to the task overview. The current task is then the first task in the overview.
- When diagnosing the current system, the task list is regenerated on paging back to the first task in the list with “--” in TSK mode.
- If a dump object was selected in dump files with multiple objects, entering F1 or “-” in INF mode cancels the selection.

5.2.2.4 The stack window (W3)

The stack window displays the contents of the first TU or TPR stack or the contents of the TU or TPR stack that was selected and marked in window W2 (PCB mode), or the contents of a TPR program manager stack (SPL mode). The remaining information depends on the mode selected in the status window (W2).

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


SYS-PCB (31BIT)    #   1       Addr: 73B001D8   TID=000402FB TSN=0VGY W3,PCB,L11
PC  : 7103B916=ETMBON1 +001D6    SVC: FA = $BOWT    LNK: 74135AE8  ISL: 7460AAE8
R00 : 00000000 =      0                 R01 : F103B803=ETMBON1 +000C3
R02 : 74692008=ETCB-2FB                 R03 : 71000800=NCTXVT  +00800
R04 : 7103C548=ETMBON1 +00E08           R05 : 73B001D8
R06 : 71247B40=ETMBOWK                  R07 : 71248A00=ETMBOWK +00EC0
R08 : 6F3D2328                          R09 : 6F3D2398
R10 : 7103B740=ETMBON1                  R11 : 7F7194C8=NMHRPUT +00E48
R12 : 80800102                          R13 : 71248A38=ETMBOWK +00EF8
R14 : F103B908=ETMBON1 +001C8           R15 : 71021974=NLCNLMAN+03234
Bourse Caller: 714477FA=ECTYP   +037FA
--------------------------------------------------------------------------------
```

Figure 21: Stack window (W3) with a PCB

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


SPL-Stack          #   1       Addr: 6F3D22D0   TID=000402FB TSN=0VGY W3,SPL,L10
ADF : 6F3D2328   (User)          ***  current   *** LNK: 6F3D2428  ADFI:  P  NYN
R00 : 74692008=ETCB-2FB                 R01 : 6F3D2398
R02 : C2404040                          R03 : FF718986=NMHRPUT +00306
R04 : 714458E8=ECTYP   +018E8           R05 : 00000001 =      1
R06 : 00000002 =      2                 R07 : 7348D600
R08 : 7A8E3A90                          R09 : 6F3D25D8
R10 : 71447748=ECTYP   +03748           R11 : 7F7194C8=NMHRPUT +00E48
R12 : 7100DA78=NLKSYSPM+014B8           R13 : 6F3D2328
R14 : F14477FA=ECTYP   +037FA           R15 : 7103B740=ETMBON1
--------------------------------------------------------------------------------
```

Figure 22: Stack window (W3) with a program manager stack

The stack window (W3) has the input fields “Stack #”, “TID”, “TSN”, “Stack select” and “Window length”.

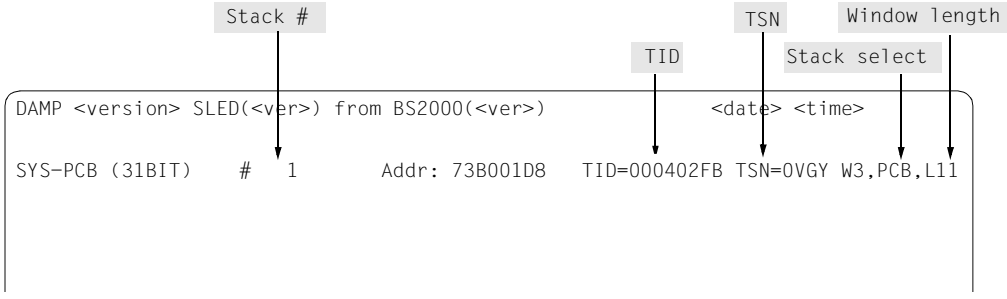


Figure 23: Input fields in the header line of the stack window

Switching from one output format to another is achieved by overwriting the entry “PCB” in the input field “Stack select” in the header line of the stack window with “SPL” or vice versa, and then hitting the DUE key.

If possible, register contents are interpreted and output as addresses or decimal values. All the fields in the output can be marked.

In order to display a memory area in one of the standard dump windows (W4-W9, W21-W99), highlight the address field and assign it an output window, see the section “Marking” on page 86.

If the access register mode flag (AR mode flag) is set in the PCB being displayed, and if the access register of the same name contains a value (ALET) other than zero, the act of marking a general-purpose register immediately assigns the corresponding data space and displays it in the requested window.
The access registers are “behind” the general-purpose registers in the stack window if the AR mode flag is set in the PCB.
If the AR mode flag is not set in the PCB, the access registers can be found in the PCB under the symbolic name ESTKARx(x=0,1,...15).

Notes

– A PCB with an x86 context is presented with all 16 registers of x86 mode. An example of this is shown in figure 24. As a /390 context can be contained in registers r12 - r15, these registers are displayed in /390 register notation (R12 - R15). All other registers have the x86-specific names.

  The contents of /390 registers R0 - R11 are not stored in x86 registers by the firmware, but in the ASSTRAN stack. These areas are also displayed in x86 mode.

  Because of the ASSTRAN optimizations, the memory areas from which DAMP obtains the contents of the /390 registers are not updated immediately each time a change occurs. Consequently the /390 registers cannot be used for reliable diagnosis in x86 mode. In this mode only the x86 registers permit reliable diagnosis!

  The type of register context shown is displayed in a special header line. You can select the context required by marking it:

  CISC /390 register in word length

  x86 x86 register in double word length

  You can also page forward and back between contexts using “>” and “<”.

  The special header line also contains the markable address of the CSA (Context Save Area). This is marked with an arrow in the left-hand margin in figure 24.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


SYS-PCB (XA  32BIT)#   1       Addr: C2E0E768   TID=00010062 TSN=SPMG W3,PCB,L19
PC  : C054B508=ETMBON3 +00548    SVC: FA = $BOWT    LNK: 00000000  ISL: 00000000
----- X86 Registers ------ CISC / X86                        CSA-Addr: C33F1000
X86 General Purpose Registers
rax: 00000000 C2724008=ETCB-062         rcx: FFFF9700 FF04186E
rdx: 00000000 00000000=     0           rbx: C0547D00 C054B202=ETMBON3 +00242
rsp: FFFF9700 FEFFFE70                  rbp: 00000000 00000000=     0
rsi: 00000000 03000000                  rdi: 00000000 00000000=     0
r8 : 00000000 BFFFE800=NCTXVT  +00800   r9 : 00000000 C32E8843
r10: FFFF9700 FE033E30=YDBADD@ +023B0   r11: 00000000 00000202=   514
R12: 00000000 80850103                  R13: 00000000 BEFFD520
R14: 00000000 C054B4F0=ETMBON3 +00530   R15: 00000000 C0062F7C=NLCNLMAN+1253C
Additional Registers in ASSTRAN Stack
R0 :          00000000=     0           R1 :          C054B202=ETMBON3 +00242
R2 :          C2724008=ETCB-062         R3 :          BFFFE800=NCTXVT  +00800
R4 :          C054D438=ETMBON3 +02478   R5 :          C2E0E768
R6 :          C054FAC0=ETMBOWK          R7 :          C30D34C0
R8 :          BEFFD520                  R9 :          BEFFD81C
R10:          C054AFC0=ETMBON3          R11:          FA70C0F0=SPMMGR  +000F0
CMD:
Key: 1=Help 2=Plk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=Dump   9=Dump
```

Figure 24: Example of a PCB stack window showing an x86 context

5.2.2.5 The dump windows (W4 - W9 and W21 - W99)

You can assign these windows various functions. By default, these windows are used as a standard dump window for the output of memory areas in edited form. This corresponds to the statement `SHOW-EDITED-INFORMATION INFORMATION=*STORAGE-EDIT, WINDOW=<w>`.

The windows W4 to W9 and W21 to W29 can be used as so-called special windows for special output. This is done using the `SHOW-EDITED-INFORMATION` statement and by specifying the intended dump window as well as the editing required. Other statements, e.g. `START-PATTERN-SEARCH`, also open a special window (see description, page 108). If no specific window is specified, DAMP positions the output in the next free window.

The occupied window is once again made available for standard output using the statement `SHOW-EDITED-INFORMATION INFORMATION=*STORAGE-EDIT`.

The following statements are supported in the W4 - W9 and W21 - W29 windows:

```
SHOW-EDITED-INFORMATION INFORMATION=*AUDIT-TABLE-EDIT

SHOW-EDITED-INFORMATION INFORMATION=*STORAGE-EDIT

SHOW-EDITED-INFORMATION INFORMATION=*DUMPED-SYSTEM-FILE

START-PATTERN-SEARCH

START-LIST-GENERATION

SHOW-EDITED-INFORMATION INFORMATION=*MEMORY-ATTRIBUTES

START-OPTION-DIALOG

START-PRODAMP-EDITOR

SHOW-EDITED-INFORMATION INFORMATION=*SUBSYSTEM-INFORMATION

SHOW-EDITED-INFORMATION INFORMATION=*TASK-TABLES

SHOW-EDITED-INFORMATION INFORMATION=*TRACE-TABLE-EDIT
```

The assignment of a function to a dialog window can also be achieved with the following abbreviated format:

```
ATT[ACH]  window#, function
```

where `window#` specifies the desired window number (4...9, 21...99).

The following entries are supported for `function`:

`AUDI, AUDIT`	Show information on AUDIT tables
`DUMP`	Restore standard dump window
`FILE`	Information on system files / sections
`FIND`	Find strings
`LIST`	Generate and print listings
`MEMA, MEMATTR`	Display memory attributes
`OPTS, OPTIONS`	Modify user options
`PROC, PRODAMP`	Use PRODAMP procedures
`SUSY`	Display information on subsystems
`TABL, TABLE`	Show tables of task-specific values
`TRAC, TRACE`	Show system trace table

Examples

`ATT 4,DUMP` or `ATT 99,FIND`

Use of the W4 - W9 and W21 - W99 windows as standard dump windows

The standard dump windows can display memory segments of the diagnosis object in dump format, hexadecimal format, character format, Assembler format or symbolic format. With the exception of character and Assembler formats, you can display memory segments in one of the standard dump windows by marking address fields and assigning them output windows, see the section "Marking" on page 86.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


ASAFNAT                          +00000=7F2774C0 SYS=000402FB          W4,D  ,L19
7F2774C0 (0000): 05A058F0  A36E0DEF  5510A372  4770A018   <==>  ???0t>????t?????
7F2774D0 (0010): 58F0A376  0DEF47F0  A0345510  A37A4770   <==>  ?0t????0????t:??
7F2774E0 (0020): A0340DE0  47F0E008  7103F264  41A0E012   <==>  ?????0?????2????
7F2774F0 (0030): 58E0E004  07FE0700  4110A03E  47F0A04E   <==>  ?????????????0?+
7F277500 (0040): 00061202  FFFFFFFF  02C9C4C1  F0F1F5F7   <==>  ????~~~~?IDA0157
7F277510 (0050): 0AED0000  00000000  05A05860  23449640   <==>  ??????????-??o
```

Figure 25: Output of an area in dump format

In the case of objects from x86 servers you can have addresses which are displayed in Little Endian format in the x86-HSI converted by marking and pressing F4 before outputting the associated memory area of DAMP in BS2000 address format (Big Endian, see page 87).

5.2.2.6 Input fields of a standard dump window (W4 - W9 and W21 - W99)

The input fields in the header line of the standard dump window are the “Symbolic address”, “Relative address”, “Absolute address”, “ASEL”, “ASID”, “Output format” and “Window length”.

In the special windows activated with the SHOW-EDITED-INFORMATION statement and the FIND, LIST, OPTS or PRODAMP window, additional or different types of input are possible (see page 108).

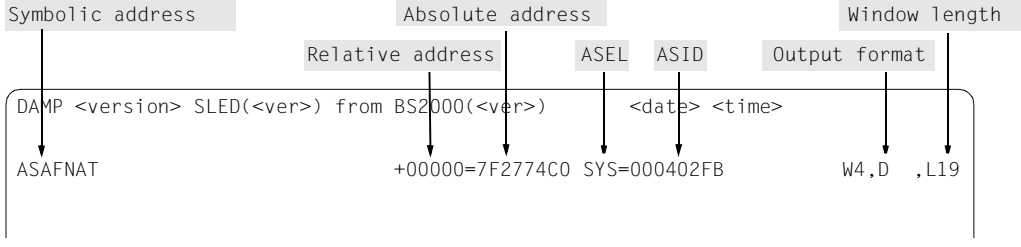


Figure 26: Input fields in the header line of the standard dump window

Symbolic address

Module names, control block names and control block field names can be entered in the input field “Symbolic address” (see figure 26).
By default, the output always shows the module name. If the memory area cannot be allocated to any module, blanks are displayed.
If a control block (field) name is displayed (CBA/CBM output format), the module name can be shown by entering “NAM” in the “Output format” field.
The names of CSECTs in subsystems or in a loaded user program may also be specified. If the current task is connected to the nonprivileged subsystem or if the CSECT is contained in the user program, the memory area is localized automatically. If privileged subsystems are loaded, the memory area is localized by DAMP even if the task is not connected.
If the module-relative display is not desired, it can be deactivated by entering “ALT” in the “ASEL” field. DAMP then relates all addresses to the current start address in this window. Furthermore, in this format, output extends beyond the module boundary.

In order to indicate that the module-relative display has been switched off, the module name is deleted.
This output format is retained during paging and if a relative or absolute address is entered.

Relative address

The current displacement between the start of the module and “Absolute address” is displayed. If the relative address is changed within the CSECT/control block area, only the relative address and the start address of the window contents change (“Absolute address”). If, however, this displacement exceeds the boundaries of the module area, the relative and absolute address outputs are adapted automatically (this also occurs when paging in the dump window).

Absolute address

The input field “Absolute address” displays the start address of the current contents of the window.

Output format

The following output formats are available:

D Dump format (default value or value after “D” is entered)
Each screen line displays 16 bytes in both hexadecimal and printable form.
The 4 word fields in hexadecimal format can be marked.

HEX Hexadecimal format (after “H” is entered)
Each screen line displays 32 bytes in hexadecimal form. All 8 word fields can be marked.

CHR Character format (after “C” is entered)
Each screen line displays 64 bytes in the form of printable characters; non-printable characters are displayed as smudges. By setting the user option “Trash character” (see page 133), any other printable character can be selected instead of the smudge).

ASS Instruction display (after “A” is entered).
Each line displays one instruction both in “disassembled” form and in machine code.
In the case of x86 objects, DAMP selects the disassembly mode in accordance with the processor mode of the CSECT (PMODE byte).
If this results in meaningless outputs, the CAS or XAS mode can be set explicitly.
In the output, ASS indicates that the /390 disassembly was used.

CAS Input: the /390 disassembly is to be used
(ASS then appears in the output).

XAS Input: the x86 disassembly is to be used.
Output: the x86 disassembly was used.

CBA Symbolic format with automatic localization of the control block (control block automatic).

CBM Symbolic format with manual localization of the control block (control block manual).

NAM Symbolic format where the name of the module containing the control block is displayed in the symbol field in place of the name of the control block.

Window length

The input field “Window length” displays the current window length of the window, including the header line. The separator line which results from a user option is not included in the displayed window size. Inputs greater than 19 are reduced to the maximum permissible size of 19 lines.

The program keys [P10] to [P12] (9750 Data Display Terminal) can be used to position the cursor to the input fields “Window size” of the following diagnostic windows, provided that at least three windows are open on screen:

[P10] positions on the “Window length” field of the first diagnostic window on screen

[P11] positions on the “Window length” field of the second diagnostic window on screen

[P12] positions on the “Window size” field of the third diagnostic window on screen

If only two windows are displayed on the screen, the [P12] key positions the cursor on the command line, and if only one window is open, the [P11] key also positions the cursor on the command line.

ASEL and ASID

The memory areas shown in the standard dump windows can be sections of the following areas:

– a virtual address space
– the main memory area (real/absolute addresses)
– the hardware system area (HSA)
– a protected processor status (PSS)
– data spaces
– dumpfile sections (SCT).

The data spaces exist alongside the task-specific and system-specific virtual address spaces and thus represent, in effect, a duplication of the virtual address spaces.

In DAMP, even a task or system-specific address space can be set as a data space. In such cases, DAMP omits the module-specific qualification.

Addressing data spaces

In addition to the 16 general-purpose registers, each process has a further 16 access registers. Depending on the way in which a particular option is set (AR mode), these access registers (with the exception of register 0) are also used for addressing memory areas.

The access register with the same name as the base register is used to address a data space if it contains any value other than zero. This data space can be up to 2 Gbytes in size and is addressed in the normal way using the base register, index register and offset.

For addressing purposes, the access registers are given an ALET (access list entry token) which uniquely identifies the data space for an address space (task or system address space). Throughout the system, the data space is identified uniquely by the SPID (space identification).

The names ASEL (Address Space Selector) and ASID (Address Space Identifier) refer to fields which can contain the following symbols:

ASEL	ASID	Meaning of the symbols
TID	<tid> (hexadecimal)	The address space is a user address space specified by its <tid> (task identifier).
TSN	<tsn> (string)	The address space is a user address space specified by its <tsn> (task sequence number).
SYS	ignored	The address space is the system address space.
ALT	<alet>-<tid> (hexadecimal)	The address space is a data space identified by <alet> (plus <tid> for user data spaces).
SPI	<space-id> (hexadecimal)	The address space is a data space identified by the (system-wide) <space-id> .

ASEL	ASID	Meaning of the symbols
RM	<segm> (hexadecimal)	The address space is the real main memory in the selected object. <segm> identifies the 4GB segment (0, 1, ...) in which the address is located.
ABS	<segm> (hexadecimal)	The address space is the absolute main memory in the selected object. <segm> identifies the 4GB segment (0, 1, ...) in which the address is located (only for complete VM2000 SLED files).
PSS	<processor> (hexadecimal)	The address space is the processor save area of the specified processor.
HSA	ignored	The address space is the hardware system area.
SCT	<section-name> (string)	The address space is a dumpfile section identified by <section-name>.

The fields “ASEL” and “ASID” are highlighted and can be overwritten.

If ALET is specified together with TID, the TID must be appended to ALET with a hyphen. If you want to set a task- or system-specific address space as the data space, you only need to enter “ALT” as “ASEL” (plus <tid> as “ASID”).

By default, the TID is displayed in the “ASID” field for areas in the user address space. If you want to have the TSN displayed, you must enter it in the “ASEL” field.

The “ASEL” field is switched to SYS when areas in the user address space are output. The TID, which is still assigned to the window, continues to be displayed in the “ASID” field. This TID can be modified as before by entering “TID” in the “ASEL” field and <tid> in the “ASID” field.

In the case of a complete VM2000 SLED, entering “ABS” - beginning at the hypervisor - causes absolute addressing to be performed regardless of whether a VM was selected. With the input “RM”, addressing is performed within a selected VM.

Abbreviated entries

All entries in the “ASEL” field can be abbreviated as desired as long as they remain unambiguous.
The TID can always be entered in abbreviated form in the “ASID” field, provided the relevant task is uniquely identified. As a rule, the last four digits of the TID are sufficient for this purpose. The TID is always output in its entirety.

Examples of entries in the “Symbolic address” and “Output format” fields

This section describes a number of important applications by providing examples of various combinations of entries in the “Symbolic address” and “Output format” fields.
The entries in the “Output format” field can be abbreviated, provided they remain unambiguous.

- Localizing a control block which can be found automatically (e.g. EXVT)

  Input: Control block name in the “Symbolic address” field

  Output: Control block name in the “Symbolic address” field
  CBA in the “Output format” field

- Localizing a control block which can be found manually

  Assuming: The memory area is already set

  Input: Control block name in the “Symbolic address” field

  Output: Control block name in the “Symbolic address” field
  CBM in the “Output format” field

- Localizing a field in the control block currently displayed

  Assuming: The control block name is already set in the window

  Input: Field name in the “Symbolic address” field

  Output: Control block name in the “Symbolic address” field
  CBM or CBA in the “Output format” field

- Localizing a field in an “automatic” control block

  Input: Field name in the “Symbolic address” field

  Output: Control block name in the “symbolic address” field
  CBA in the “Output format” field

- Localizing an “automatic” control block manually

  Assuming: The memory area is already set.

  Input: Control block name in the “Symbolic address” field
  CBM in the “Output format” field or input in the “Absolute address” field

  Output: Control block name in the “Symbolic address” field
  CBM in the “Output format” field

- Overlaying an area with a control block as of a field name

  Assuming: The memory area is already set

  Input: Field name in the “Symbolic address” field
  CBM in the “Output format” field

  Output: Control block name in the “Symbolic address” field
  CBM in the “Output format” field

- Overlaying an area with a control block as of a relative address

  Assuming: The memory area is already set

  Input: Control block name in the “Symbolic address” field
  Relative address (relative to start of control block)

  Output: Control block name in the “Symbolic address” field
  CBM in the “Output format” field

- Displaying the module in which a control block is located

  Input: NAM in the “Output format” field

  Output: Module name in the “Symbolic address” field
  NAM in the “Output format” field

- Switching from symbolic representation to dump format

  Input: D in the “Output format” field

  Output: Module name in the “Symbolic address” field
  D in the “Output format” field

- Displaying an area in a module

  Input: Module name (plus relative address if required) in the “Symbolic address” field

  Output: Module name in the “Symbolic address” field
  Display in the “Output format” field is retained; CBA, CBM or NAM is changed to D

- Switching from dump format to disassembled format

  Input: ASS in the “Output format” field

  Output: Module name in the “Symbolic address” field
  ASS in the “Output format” field

5.3 Operation

5.3.1 Basic functions

5.3.1.1 Calling DAMP

DAMP is called in the system on which the diagnosis object exists with **/START-DAMP.** DAMP is then started from the user ID saved under IMON. Depending on the selected function, the appropriate processing modules are loaded dynamically from the set module library by the load program.

If DAMP was not installed on a system with the default installation or if DAMP is to be started with special user options (see page 133), **/START-EXECUTABLE-PROGRAM** must be used with the name of the possibly modified DAMP load phase (delivery name: SYSPRG.DAMP.<ver>).

For analysis in the current system, the calling user ID must have the read test privilege 8. The privilege must be activated beforehand with the MODIFY-TEST-OPTIONS command.

When DAMP is called **in interactive mode**, functions are automatically assigned to the programmable keys and the DAMP screen is then displayed with the HELP window active (this does not apply to batch and procedure modes).

DAMP can also be started as a **batch job** (see page 165). In this case, the entries are read from SYSDTA and processed as if they were entered in the command line. This function is primarily intended for generating lists.

DAMP can also be started in **procedure mode** (see page 165). This function is particularly suited for a standard preliminary “on-screen diagnosis run”.

5.3.1.2 Controlling program execution

Program execution is controlled by the automatically programmed P keys, the K keys, the F keys, by marking fields with the MAR key and by entering suitable statements.

5.3.1.3 Assigning and opening the diagnosis object

The object to be diagnosed (dump file or system) can be assigned in one of two ways:

– The statement `OPEN-DIAGNOSIS-OBJECT dumpfilename` assigns a SLED, SNAP, system, user or area dump for diagnosis. It is advisable to assign the link names #0,#1,...,#9 to these files before or during program execution, since this permits the assignment to be abbreviated as follows:

```
OPEN-DIAGNOSIS-OBJECT #n (n = 0,...,9)
```

Partially qualified file names and wildcards may be used in the file name. If a file is uniquely identified by the partial qualification or wildcard, DAMP opens this file.

The KIND-OF-OBJECT parameter can be used to specify whether the dump object is to be opened as a BS2000 object, a SELF-LOADER or a PAM file. For more information, see also the `OPEN-DIAGNOSIS-OBJECT` statement on page 197.

The statement `OPEN-DIAGNOSIS-OBJECT *SYSTEM` is used to assign the active system as the object to be diagnosed. Note, however, that the calling user ID must have the read test privilege 8 for this purpose.

– The list mask can also be used to select, assign and open a dump file, even if no list is to be printed out. It is also possible to enter a partially qualified file name or a file name containing wildcards in a field provided for this purpose and then to select a file from the list of matching file names found by the system. For further details, see section “Generating and printing lists (special window: LIST)” on page 147.

5.3.1.4 Modifying the diagnostic windows

Since the diagnostic windows are displayed in the diagnostic area of the screen, they cannot be longer than this area, i.e. the maximum length of a window is 19 lines. However, the diagnostic windows can be shortened if, for example, several windows are to be displayed simultaneously in the diagnostic area. The minimum length of a diagnostic window is 2 lines.

The order in which the windows are displayed, their length, their contents and their output formats can be controlled by:

– the MODIFY-SCREEN-LAYOUT statement

– the program keys P1 to P9 or by entering the window number (0...9, 21...99) on the command line

– marking address fields, keywords and certain output lines (task line, PCB line, hit lines, etc.)

– paging forward and backward with --, ++, -, +, -n, +n and F3 or F1 or via the corresponding program keys (see the section on “Paging in a diagnostic window” on page 86)

– making entries in the input fields in the header lines

– the SHOW-EDITED-INFORMATION, START-PATTERN-SEARCH, START-LIST-GENERATION, START-OPTION-DIALOG, START-PRODAMP-EDITOR statements.

When the program is started, the diagnostic windows contain their default values:

– The default length for a diagnostic window is 19 lines, except for windows W2 and W3, where the window length is matched to the actual amount of information to be displayed.

– The output format for the dump windows (W4 - W9 and W21 - W99) is, by default, the dump format (display D). These windows are flagged as unused (empty).

– Output in the “RM” or “ABS” dump format is used if virtual addressing is not possible. The page “0” is then displayed.

After DAMP is called, the HELP window (W1) is displayed with an overview of the help chapters.

If the diagnosis object to be processed is changed by means of an OPEN-DIAGNOSIS-OBJECT statement, all assignments previously set for the diagnostic windows are reset. Only the settings for the window length and output format are retained. The LIST and PRODAMP windows are an exception; they are retained even if the dump object is changed.

Restoring the screen contents

If a line fault or an operator message causes the screen display to be displaced, the original display can be restored by means of hitting K3.

The MODIFY-SCREEN-LAYOUT statement

The MODIFY-SCREEN-LAYOUT statement moves the specified window(s) to the beginning of the window display sequence and optionally defines a new window size (see page 195).

Program keys P1 - P15

Program keys P1 to P9 move the corresponding windows W1 to W9 to the beginning of the window display sequence and display them in the diagnostic area. After you have marked an address field the associated memory area is assigned to the diagnostic window which is selected via keys P1 to P9.

Program keys P10 to P12 can be used to position the cursor to the input field “Window size” of the first three diagnostic windows displayed. If less than three windows are displayed, P12 and P11 position the cursor to the command line.

If you transfer an entry using program key P13, DAMP passes it directly to any PRODAMP procedure which may be active. This allows you to write your own user interface for PRODAMP applications.

You can use program keys P14 and P15 to scroll in the DAMP message history. P14 enables you to scroll back in the message lines (lines 2 and 3) and P15 to scroll forward.

The program keys P1 to P15 are automatically loaded when DAMP is started. If the settings for the P keys have been lost (e.g. after a task switch via OMNIS), they can be reset by interrupting the program by hitting the K2 key and then entering the /RESUME-PROGRAM command.

On data display terminals which incorporate the “read P area” function (as of 9762 Data Display Terminals), DAMP saves the current contents of the program keys before loading them with the new values if **Save P-Keys = yes** is set in the DAMP options (see page 135). Every time that DAMP is interrupted using K2 or when DAMP is terminated, the contents of the program keys most recently saved are reloaded. Any changes to the contents of the function keys made while DAMP is suspended are saved when the /RESUME-PROGRAM command is issued.

Alternative to program keys P1 - P9 and P13

Instead of using the program keys P1 through P9 and P13 it is possible to enter the appropriate number (1 through 9 or 13) in the command line and activate by pressing the DUE key.

Paging in a diagnostic window

Function keys F3 and F1 can be used to page forwards and backwards within a diagnostic window. This function of F1 and F3 is also mapped onto the last two program keys: on a terminal with 20 program keys (9750) P19 is equivalent to F1 and P20 to F3.

In general, the use of +, -, ++, --, +n, -n for paging is also supported:

+ page forward by one window

- page backward by one window

++ page to end of list, chapter or module

-- page to beginning of list, chapter or module

+n page forward n lines

-n page backward n lines

If two or more windows are displayed on the screen, paging occurs in the most recently used window. Paging with F1 or "-" initially moves back only to the start of the current module, control block or chapter. This boundary can then be crossed by entering another paging command.

Marking

The contents of standard dump windows are generally a portion of the system or user memory which contains command, data and address fields. Marking an address field, i.e. positioning the cursor to this field and pressing the MAR key, assigns this address as the start address of the current dump window or of a new one. Similarly, addresses can be marked in the stack window (W3) and various information can be marked in function-specific special windows (FIND, AUDIT, etc.).

A marked memory location is as far as possible not displayed in the window which contains the marking.
To assign a particular standard dump window to a marked address, after you have marked it press one of the program keys P4 through P9 (for output in one of the windows, W4 through W9) or enter the window number (4..9, 21..99) in the command line and then press DUE. Standard dump windows can also be assigned using the MODIFY-SCREEN-LAYOUT statement.
If you select no dump window after entering marking, i.e. only press DUE, DAMP selects the next standard dump window which is displayed in the diagnosis area. If no further dump window is available in the diagnosis area, DAMP selects the dump window which is currently set as standard.

Depending on the number of windows displayed in the diagnosis area, the display can take place in up to six standard dump windows simultaneously. If more fields are marked, the surplus fields are ignored.

Highlighted keywords in the help window (W1) and the task or PCB lines in the status window (W2) can also be marked. Hitting the DUE key then provides more detailed information on the marked term or the PCB.

In standard dump windows, the first column in each line can be marked. The line marked in this manner is positioned to the top of the window in the next output.

Inadvertently marked fields can be cleared by hitting the MAR key again.

Converting the address display for contexts in the x86-HSI

On x86 servers memory addresses in x86 contexts exist in “Little Endian” format. BS2000 uses”Big Endian” format to display addresses. During ongoing operation the firmware converts the addresses to the correct format at the HSI transition.

When editing BS2000 tables, DAMP automatically displays the addresses in the Big Endian format of BS2000 in special windows (e.g. when editing a PCB with x86 context in W3).

However, when a memory area with x86 context is displayed in a standard dump window, the specified area is displayed unchanged by DAMP. This is also the case when a DSECT is applied to the area. You can have an address converted before the associated memory area is displayed by DAMP by marking it and pressing F4. When a standard dump window is specified as the output window, the area from the converted address is output to the selected window. However, the address display at the marked position is **not** converted.

The possible output windows are described in the section “Marking” on page 86.

F4 is only effective for x86-HSI objects.

5.3.1.5 Interrupting and resuming DAMP operation

The K2 key can be used at any time to interrupt execution of DAMP and to switch to the command line, where any system command may be entered.

Control is returned to the DAMP program by means of the RESUME-PROGRAM command. You can, however, also resume working with DAMP from the command level, namely via the STXIT interrupt routine and the INFORM-PROGRAM command (see section “System level” on page 224).

5.3.1.6 Terminating DAMP

DAMP can be terminated in a number of ways:

– by entering the **END** statement in the command line

– by pressing the K1 key. Confirmation is only required if the appropriate user option has been set (see page 133).

Alternatively, DAMP can also be terminated as follows:

– Press the K2 key (which interrupts the current program and switches to the command level) and enter the INFORM-PROGRAM command with one of the following texts:

/INFORM-PROGRAM MSG='*END' /INFORM-PROGRAM MSG='*HALT' /INFORM-PROGRAM MSG='*TERMINATE'	Terminate without a dump
/INFORM-PROGRAM MSG='*DUMP' /INFORM-PROGRAM MSG='*TERMD'	Terminate with a dump

5.3.2 Output of dump data

Since every software problem is essentially unique, this manual cannot offer patent solutions for all of them. It can only tell you what information can be found in a diagnosis object and how to find it. It is up to you to decide what you should be looking for.

Diagnosis is initiated by calling the DAMP program. The help window W1 is displayed with an overview of the help chapters.

What happens next is controlled by means of DAMP statements, function keys, modifying input fields and marking fields.

It is a good idea to activate logging of the diagnosis run so that it is possible for the same or different diagnostic technicians to reconstruct, at a later time, diagnostic actions taken.

5.3.2.1 Automatic interpretation of the output data

During display of the status and stack windows and during output of a memory segment in symbolic format (overlaid with a DSECT), DAMP attempts to interpret the contents of the various fields. In particular, it attempts to relocate addresses, i.e. to represent them as a module and a displacement within this module. This is done in accordance with the following rules.

In dump windows,
the start address is interpreted relative to a specific module. Furthermore, all displayed relative addresses refer to the beginning of the module visible in the window. The contents are edited, at most, up to the end of the module.

If this form is not desired (when, for example, the relative addresses are to refer to the start of a table located within a module), the following steps must be carried out:

– position to the start of the table

– enter “ALT” in the ASEL field.

DAMP then uses the current start address of the window as the basis for address relocation. In this case, output is also continued past the end of the module.

All addresses in diagnosis objects are always interpreted as 31-bit addresses (/390 objects) or 32-bit addresses (x86 objects). If this is not desired, you will need to overlay the memory segment with a DSECT and then proceed as in the case of symbolically edited segments (see section “Symbolic layout” on page 97).

In TU-PCBs,
the addresses are interpreted as 24-bit, 31-bit or 32-bit addresses, depending on the addressing mode set in the PCB. 32-bit addresses are only possible for x86 objects. Modules from the connected nonprivileged subsystems and CSECTs of any loaded user program are taken into account for address relocation. General-purpose registers 0 and 1 are never addressed relatively.

If the access register mode flag (AR mode flag) is set in the PCB being displayed, and if the access register of the same name contains a value (ALET) other than zero, the act of marking a general-purpose register immediately assigns the corresponding data space and displays it in the requested window.

In TPR-PCBs,
addresses are always interpreted as 31-bit addresses (/390 objects) or 32-bit addresses (x86 objects). Modules of the Control Program (CP) and of all privileged subsystems thar are loaded are taken into account for address relocation.

The same applies to TPR PCBs where the AR mode flag is set as described above for TU PCBs.

In symbolically edited memory segments,
addresses are always interpreted as 31-bit addresses (/390 objects) or 32-bit addresses (x86 objects).
In special cases (for example, when viewing user parameter lists), this may not be desired. In such cases, the statement

```
MODIFY-OBJECT-ASSUMPTIONS ADDRESSING-MODE=*PAR(<control-block>, *NXS/ *XS31)
```

can be used to specify that the addresses are to be regarded as 24-bit addresses for the specified control block. Depending on whether the data is in user memory or in system memory, all modules or only modules belonging to the Control Program and to class 4 subsystems are taken into account for address relocation.

In function-specific windows
such as the TRACE, FIND, SUSY or TABL window, addresses are always interpreted in the same way as for memory segments.

Exception

Address relocation is performed in the same way as for TU-PCBs only in the case of AUDIT windows for a TU AUDIT.

5.3.2.2 Output of status information

If you open one of the diagnostic objects SLED, SNAP dump or active system, or if you enter TSK mode in the status window (W2) after opening a system dump, you obtain the status information on the tasks which occur. The amount of data to be displayed will generally exceed the maximum permitted window length, but the remainder of the data can be displayed by paging.

Marking a task line or entering the name of the desired task in the appropriate input field in the header line and hitting the DUE key causes information on all PCBs of this task (PLK mode) to be displayed. If the entered TSN is shorter than 4 characters, it is extended by leading zeros to a length of 4 characters. If a TSN begins with blanks (e.g. in system tasks), then these must be entered.

For PCB output, the size of the status window (W2) is implicitly set to the number of lines needed to display the actual information. When the first PCB is displayed, the F1 key returns you to the task overview.

In the case of system, area and user dumps, the PLK mode is immediately set when the dump file is opened, i.e. the list of PCBs of the task which caused the dump is displayed.

The TPR program manager stack can be displayed by entering “SLK” in the “Mode select” field.

In INF mode, the object currently open is always described. If, in the case of dump files with a number of objects, selection is possible, it is done by marking the required objects. This selection can be undone by setting the INF mode in the status window and pressing the F1 key.

You can always switch between the different modes (INF, TSK, PLK, SLK) of the status window by making entries in the “Mode select” field.

If a PCB is marked, the stack for this PCB is displayed in the stack window (W3). If no PCB is marked when the stack window is activated by means of either P3 or a `MODIFY-SCREEN-LAYOUT` statement, the stack for the first PCB is displayed.

5.3.2.3 Output of stack contents

The contents of program control blocks (PCB) are displayed in the stack window (W3). Every address field contained in the stack window can be marked.

The following applies to **system, user or area dumps**:
In order to view the first PCB of this task, you only need to press the P3 or DUE key. This causes the stack window (W3) to be displayed with the contents of the first PCB (default value).
If you want to view the contents of another PCB, first press either the P2 or DUE key. This produces the status window (W2) containing the PCB list. Mark the line with the desired PCB and then press the P3 key. The stack window (W3) with the contents of the marked PCB will now be displayed.

If a SLED or SNAP is available, a task must first be selected. Task 1 is set by default. You select a task by marking a task line in the status window (W2) and sending it off with DUE. This causes the PCB list for the marked task to be displayed in the status window. Marking the line containing the desired PCB and then hitting the P3 causes the stack window (W3) with the contents of the marked PCB to be displayed.

The data to which the registers of the PCB refer can be displayed by marking the program counter or an address field in the PCB registers. Subsequently hitting one of the program keys P4 to P9 assigns the marked address to a dump window. Alternatively the dump window can be selected using the MODIFY-SCREEN-LAYOUT statement or by specifying the window number (4...9, 21...99) in the command line and transferring it with DUE.

Up to six addresses can be marked and assigned to dump windows at any one time. To do this, the MODIFY-SCREEN-LAYOUT statement must first be entered, e.g. as follows:

```
MODIFY-SCREEN-LAYOUT FIRST=3(SIZE=10), SEC=4(SIZE=2), THIRD=5(SIZE=2),
                     FOURTH=6(SIZE=2)
```

This causes the windows to be displayed in the specified order and with the specified sizes. Subsequently marking three address fields of the PCB and hitting the DUE key causes the marked addresses to be assigned to dump windows W4, W5 and W6.

If you want to output the TPR program manager information instead of the PCB information, enter

– “SLK” in the “Mode select” input field of the status window (W2) (see page 61) or

– “SPL” in the “Stack select” input field of the stack window (W3).

5.3.2.4 Output of system tables

DAMP can automatically localize the following system tables in the assigned diagnosis object and display them in “symbolic format” (in accordance with their DSECT layout) in a dump window

Name	Abbreviation	DSECT name
Executive Vector Table	XVT	EXVT
Task Control Block	TCB	ETCB
Job Control Block	JCB	EJCB
System Virtual Memory Table	SVMT	EVSMT
User Virtual Memory Table	UVMT	EVUMT

Table 5: System tables that can be automatically localized

Entering the DSECT name in the input field “Symbolic address” of a dump window (W4 - W9 or W21 - W99) causes the memory segment to be localized and displayed from the beginning of the table in symbolically edited form, i.e. with the displacement address, the DSECT field names, the field contents and with a possible interpretation in accordance with the field definition. In the case of the task-specific tables TCB, JCB, and UVMT, it may also be necessary to enter the desired task when processing SLED and SNAP files. Otherwise, the Task 1 or the last task which was selected is used as the default value.

```
DAMP <version> SYSDUMP(<ver>) from BS2000(<ver>)    <date> <time>


EVSMT                            +00000=711A5000 TID=000A00CB          W4,CBA,L19
000 EVSMID  : E2E5D4E3                    | 004 EVSMVER : 00F4F6F0
008         : 0000000000000000            | 010 EVSMRASZ: 00010000 = 65536
014 EVSMRASM: 00010000 = 65536            | 018 EVSMRASI: 00010000 = 65536
01C EVSMMMLM: 0000FFFF = 65535            | 020 EVSMHFR#: 0000FFFF = 65535
024 EVSM#AFR: 00010000 = 65536            | 028 EVSMC1FR: 000003F2 =  1010
02C EVSMMBFR: 00000000 =     0            | 030 EVSMPPTP: 710E0100 = EMMENTAL+40100
034 EVSMPPTE: 710E0900 = EMMENTAL+40900
038 EVSMPPTA: 710E0900 = EMMENTAL+40900
03C EVSMPPAT: 710E0000 = EMMENTAL+40000
040 EVSMPPAE: 710E0040 = EMMENTAL+40040
044         : 00000000                    | 048 EVSMPPID: 0000000080000600
050 EVSMPPTI: 02                          | 051         : 00
052 EVSMPPSH: 000A     =    10            | 054 EVSMPPM1: 0FFFE000
058 EVSMPPM2: 00001FFF =  8191            | 05C EVSMPPRS: 0100     =   256
05E EVSMPPVS: 2000                        | 060 EVSMPPES: 0020     =    32
062         : 0000                        | 064 EVSMPPTR: 000E0900
068         : 00000000                    | 06C EVSMPAGP: 0000D149 = 53577
070 EVSM#CFR: 0000D0E8 = 53480            | 074 EVSM#CFB: 0000D0E8 = 53480
CMD:
Key: 1=Help 2=Plk 3=PCB 4=EVSMT  5=Dump   6=Dump   7=Dump   8=Dump   9=Dump
```

Figure 27: Output of the SVMT in symbolic format; display in the key line: EVSMT

Instead of one of the DSECT names mentioned in table 5, it is also possible to enter a field name from the DSECTs. The output then begins with the corresponding memory address.

The F3 and F1 keys or +/-/++/-- can be used to page forwards and backwards within the DSECT, but they cannot be used to leave the DSECT area. Leaving the DSECT area can be done only by switching to another layout in the “Output format” field.

Other system tables are not localized automatically. The person carrying out the diagnosis must find them via address chaining. Symbolic output of these tables can be effected by overlaying them with a DSECT from the symbol file (see section “Symbolic layout” on page 97).

“Displaced” overlaying is also possible. In this case, the memory segment is edited only as of the specified DSECT field name, rather than in the format of the entire DSECT.

5.3.2.5 Output of processor storage areas

The **local XVTs** for multiprocessors can be displayed as follows:

– After opening a SLED or SNAP dump, enter the field name EXVTLMD1 in the “Symbolic address” input field of a dump window and press DUE. This causes the global XVT to be displayed in this window.

– Set up two free windows with the aid of the `MODIFY-SCREEN-LAYOUT` statement.

– Mark the addresses in the fields EXVTLMD1, EXVTLMD2 etc. and press DUE. This causes the local XVTs to be displayed in the new windows in dump format.

– You can overlay the memory segments with the format of the EXVT, starting at field EXPROCAR (beginning of the local part).
Do this by entering the field name EXPROCAR in the input field “Symbolic address” and entering the keyword “CBM” in the “Output format” input field.

DAMP offers the following alternative options for simplification purposes:

– Enter the “pseudo” CSECT EXVT-XXX in the “Symbolic address” field and press DUE. XXX is the 3-digit hexadecimal number of the required logical machine.

– Replace EXVT-XXX with EXVT or EXPROCAR **and** enter the **“**CBM” key word in the “Output format” field. Then press DUE.

Example

Output of the local XVTs for multiprocessors

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


EXVT                            +00CE8=710014E8 SYS=00010001          W4,CBA,L 3
CE8 EXVTLMD1 : 71F9D800 = EXVT-001     | CEC EXVTLP02 : 71F9E800 = EXVT-002
CF0 EXVTLP03 : 71F9F800 = EXVT-003     | CF4 EXVTLP04 : 71FA0800 = EXVT-004
-------------------------------------------------------------------------------
EXVT                            +00000=71F9D800 SYS=00010001          W5,CBM,L11
000 EXPROCAR : E7E5E3F4                | 004          : 00000000
008 EXVTTSCE : 0000000000000000        | 010 EXVTCWKA : 00000000 =     0
014          : 00000000000000000000000000000000000000000000000000000000000000000000
034          : 00000000000000000000000000000000000000000000000000000000000000000000
054          : 00000000000000000000000000000000000000000000000000000000000000000000
074          : 00000000000000000000
080 EXVTTSPD : 7111E000 = EMMINIT1+3A000
084 EXVTTSTD : 0010307F                | 088 EXVTTST@ : 71103000 = EMMINIT1+1F00
08C EXVTASTO : 0011F040                | 090 EXVTSSPD : 00000000 =     0
094 EXVTSSTD : 00000000 =     0        | 098 EXVTSST@ : 00000000 =     0
-------------------------------------------------------------------------------
EXVT                            +00000=71F9E800 SYS=00010001          W6,CBM,L11
000 EXPROCAR : E7E5E3F4                | 004          : 00000000
008 EXVTTSCE : 0000000000000000        | 010 EXVTCWKA : 00000000 =     0
OMD:
Key: 1=Help 2=Tsk 3=PCB 4=EXVT   5=EXVT   6=EXVT   7=Dump   8=Dump   9=Dump
```

Figure 28: Output of the local XVT in symbolic format

In addition, you can automatically locate a processor-local **processor storage area** (PSA) by entering the “pseudo” CSECT PSA-XXX.

5.3.2.6 Output of hardware information

In SLED files, DAMP can localize the hardware information and display it in a diagnostic window.

Entering the appropriate keyword in the input field “ASEL” and any parameter required in the input field “ASID” in one of the dump windows W4 - W9 or W21 - W99 causes DAMP to localize the corresponding memory segment and to display it in the selected output format in this dump window.
The following entries are permitted:

- in ASEL: „HSA“
  in ASID: input ignored
- in ASEL: “PSS”;
  in ASID: <processor-number>(hexadecimal)

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


                                +00000=  000000 PSS=0                 W4,D  ,L19
00000000 (00000):E2E3C1E3  0600E500  00000000  00000000   <==>  STATV
00000010 (00010):00000000  00000000  00000000  00000000   <==>
00000020 (00020):31020001  21600000  DC400000  00000000   <==>  -
00000030 (00030):00000000  00000000  00000000  00000000   <==>
00000040 (00040):0125C000  00280000  040A0000  80FFFFFF   <==>  ~~~
00000050 (00050):FFFFFFFE  78F9AB00  FFFFFFFF  FFFFF000   <==>  ~~~9~~~~~~0
00000060 (00060):00000000  00000000  00000000  00000000   <==>
00000070 (00070):00000000  00000000  00000000  00000000   <==>
00000080 (00080):0CB5FC40  0040A07F  00173440  00000000   <==>    "
00000090 (00090):00000000  00120200  FF000000  0010507F   <==>  ~&"
000000A0 (000A0):00000000  00000000  00000000  00000000   <==>
000000B0 (000B0):00000000  0040A07F  FF000000  70FD8BE8   <==>   "~}Y
000000C0 (000C0):00000000  00000000  00000000  00000000   <==>
000000D0 (000D0):00000000  00000000  00000000  00000000   <==>
000000E0 (000E0):00000040  711B9011  F12663D4  711B9000   <==>   1M
000000F0 (000F0):00000000  00000000  17D78400  711B9DD8   <==>  PdQ
00000100 (00100):0003F480  7103F480  71000000  71000800   <==>  44
00000110 (00110):71266550  711B9E20  711B9000  00000000   <==>  &
CMD:
Key: 1=Help 2=Tsk 3=PCB 4=PSS-0 5=Dump   6=Dump   7=Dump   8=Dump   9=OPTS
```

Figure 29: Output of the “Processor Saved Status” in the dump window W4

5.3.2.7 Output of memory segments

Memory segments which cannot be localized automatically can be found either by marking the corresponding address fields, by explicitly entering the addresses in the input fields of a dump window, or by paging within a displayed memory segment.

The memory segment is displayed in the output format which was last selected. The output format can be changed by changing the entry in the corresponding input field in the header line. However, this also changes the amount of information displayed:

Format	Display	Input	Information per line	Information per window
Dump	D	d,D	16 bytes	288 bytes
Hexadecimal	HEX	h,H	32 bytes	576 bytes
Character	CHR	c,C	64 bytes	1152 bytes
Assembler	ASS, XAS ASS XAS	a,A cas,CAS x,X	1 statement	
Symbolic	CBA CBM NAM	cba,CBA cbm,CBM n,N	max. 32 bytes	

Table 6: Relationship between output format and the amount of information output

When a dump window is assigned for the first time, the default setting is dump format D with a window size of 19 lines.

Symbolic output cannot be selected by an entry in the relevant input field. It is automatically selected for the output of tables which can be localized and for cases where a memory segment is overlaid with a DSECT (where a DSCET is specified in the field “symbolic address”).

5.3.2.8 Symbolic layout

If a memory segment is structured in the form of a standard DSECT, then DAMP can display this segment in the format of this DSECT. (The term DSECT is used below to refer to structures from the various programming languages, e.g. for ASS-DSECTs, MODELs (SPL), STRUCTUREs (C).) The various fields of the DSECT are interpreted in accordance with their data types. Numeric values are displayed both in hexadecimal form and as an address or decimal value. Strings are always displayed as strings.

Since the definitions of the field names in the DSECTs being used do not always match their meanings (e.g. CL4 instead of AL4), some of the interpretations may be slightly inaccurate. This can be rectified by modifying the DSECTs and then generating a new or modified symbol file (see section “Using private symbol elements” on page 142).

In addition to the global and task-specific tables which can be localized automatically, any desired memory segment can be displayed in the format of a DSECT. Two different methods are available.

Overlaying with a complete DSECT

In order to overlay a memory segment with a DSECT, the beginning of the displayed memory segment must correspond to the beginning of the DSECT. Entering the DSECT name in the input field “Symbolic address” and entering the keyword “CBM” in the input field “Output format” causes the output to be displayed in symbolic format. It is not necessary to enter the keyword “CBM” if DAMP cannot automatically localize the DSECT.

If the memory segment is localized by entering an absolute address in the header line, a DSECT name can be entered at the same time. The output of the memory area then follows immediately in symbolic format.

```
DAMP <version> SYSDUMP(<ver>) from BS2000(<ver>)    <date> <time>


ESTK                              +00000=739073A8 TID=000A00CB          W5,CBM,L19
000 ESTKTBLH: D7C3C2                    | 003         : 04
004 ESTKSIND: 84                        | 005 ESTKHSI : 01
006 ESTKIND : 01                        | 007 ESTKIND1: 01
008 ESTKAFIN: 60                        | 009 ESTKEIAF: 00
00A ESTKPCBL: 01C8                      | 00C ESTKSLNK: 72F91008
010 ESTKISLK: 72F91008                  | 014 ESTKTSF1: 00
015 ESTKTSF2: 00                        | 016 ESTKTSS : 00
017         : 00                        | 018 ESTKAUDM: 00000000 =     0
01C ESTKEXRT: 713974A0 = NLMLOCK5+0CA0  | 020 ESTKCLK : 00000000 =     0
024 ESTKCCLK: 0000FFFF = 65535          | 028 ESTKSTXC: FF
029 ESTKSTXB: FF                        | 02A ESTKP1EV: EF
02B ESTKP1SE: FF                        | 02C ESTKPOST: 71E63140
030 ESTKBOID: 000A00CB                  |
034         : 000A00CBF1A941860006000500000000000000000
048 ESTKLPSC: 00000000 =     0          | 04C ESTKBRSE: 00000000 =     0
050 ESTKBRS@: 71249400 = ETMBOWK +18C0  | 054 ESTKCONF: 80
055 ESTKCLEV: EF                        | 056 ESTKSTID: 00
057 ESTKPLVL: 00                        | 058         : 0000000000000000
CMD:
Key: 1=Help 2=Plk 3=PCB 4=ESMFHD 5=ESTK   6=Dump   7=Dump   8=Dump   9=FIND
```

Figure 30: Output of a stack in symbolic format. The DSECT name is displayed in the key line.

The contents of the “Output format” field indicate whether a DSECT has been localized automatically by DAMP (field contains CBA = control block automatic) or whether the user has positioned the DSECT (field contains CBM = control block manual). If the field contains the entry CBM, this means that entering a field name of the overlaid DSECT in the input field “Symbolic address” will cause paging within the DSECT format. Automatic positioning to the field name of the corresponding system table, which may be possible here, is not active.

The assignment of a DSECT to a window is stored. As a result, if the same structure is localized in another memory segment (e.g. in the case of chained lists) and this memory segment is then assigned to this diagnostic window by, for example, marking it and pressing the appropriate P key, the window is not switched to dump layout. Instead, the output is immediately edited symbolically to match the previously saved DSECT.

This also happens if some other memory segment is assigned to the window. This segment is then displayed in the format of the saved DSECT, which will probably be incorrect. If this is the case, you will then need to either enter the name of the correct DSECT or switch to dump format by entering D in the “Output format” of the header line.

Overlaying with an offset DSECT

If a memory segment is formatted in accordance with only a part of a DSECT (as is the case for the local XVTs for multiprocessors, see figure 28 on page 95), then a displaced overlay must be used. In this case, the appropriate DSECT field name must be entered in the input field “Symbolic address” and the entry “CBM” must be made in the “Output format” field. The keyword “CBM” need not be entered if the DSECT cannot be localized automatically.

This can also be used to save having to page back to the start address of a table. If the field name of the start of the displayed memory segment is known, the displayed segment can be overlaid by specifying the field name again.

Paging within a DSECT is possible via F1 / F3 and +/-/++/--, but only as far as the beginning or end of the DSECT. The symbolic layout is not affected by this.

Lists of all DSECTs that can be specified for all supported BS2000 versions can be found on page 330. DSECTs from private symbol files can also be used (see section “Using private symbol elements” on page 142 for further details).

In the symbolic output, X'4F' is used as the column separator character. With certain terminal settings and with certain printer character sets, this character is not represented as the vertical line “|”. This can be remedied with the aid of the user parameter “Column separator (list)” (see page 135).

Overlaying with the pseudo-DSECT WORDLIST

If a memory segment is not described in the form of a DSECT which can be overlaid, but (possibly) contains address references to system areas, then these addresses can be edited symbolically with “module name + displacement” with the aid of the pseudo-DSECT WORDLIST.

For each word of the memory segment, this pseudo-DSECT assumes the Assembler declaration DS AL4. If the contents of the word formally permit, they are displayed as “module name + displacement”.

It is thus possible, for example, to overlay the constant area of a module with WORDLIST in order to implement module-relative editing of the external addresses used therein.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


WORDLIST                        +00000=AFFEA4FC TID=000101E1          W4,CBM,L19
000 WORD000 : B13DF520 = DPSUSTA +00A0 | 004 WORD001 : B13EB7B8 = DPSUSTA@+B1F8
008 WORD002 : B13DF520 = DPSUSTA +00A0 | 00C WORD003 : AFFA0000
010 WORD004 : AFFEA3C0                 | 014 WORD005 : B13EBAC8 = DPSUSTA@+B508
018 WORD006 : B13E3010 = DPSUSTA@+2A50 | 01C WORD007 : AFFEFB6E
020 WORD008 : AFFEA720                 | 024 WORD009 : B1019140 = DKCMSPL@+3B80
028 WORD010 : B10124A0 = DKCMSPL +00A0 | 02C WORD011 : B0427D80 = NLKSYSPM+1F40
030 WORD012 : AFFEA378                 | 034 WORD013 : B1019C00 = DKCMSPL@+4640
038 WORD014 : B1021258 = DKCMSPL@+BC98 | 03C WORD015 : AFFEF006
040 WORD016 : B101A040 = DKCMSPL@+4A80 | 044 WORD017 : AFFEAF48
048 WORD018 : 00000003 =     3         | 04C WORD019 : 00000001 =     1
050 WORD020 : B03FE160 = NCTXVT  +2160 | 054 WORD021 : 00000004 =     4
058 WORD022 : AFFEA3C0                 | 05C WORD023 : B13E3064 = DPSUSTA@+2AA4
060 WORD024 : B13E2AC8 = DPSUSTA@+2508 | 064 WORD025 : AFFEFB6E
068 WORD026 : AFFEA720                 | 06C WORD027 : AFFEA720
070 WORD028 : B10128B0 = DKCMSPL +04B0 | 074 WORD029 : B10226E8 = DKCMSPL@+D128
078 WORD030 : B10124A0 = DKCMSPL +00A0 | 07C WORD031 : B0427D80 = NLKSYSPM+1F40
080 WORD032 : AFFEA378                 | 084 WORD033 : B1022A54 = DKCMSPL@+D494
088 WORD034 : B1020B48 = DKCMSPL@+B588 | 08C WORD035 : AFFEF006
CMD:
Key: 1=Help 2=Plk 3=PCB 4=WORDLI 5=Dump   6=Dump   7=Dump   8=Dump   9=Dump
```

Figure 31: Editing with the DSECT WORDLIST; display in the key line: WORDLIST

In the case of class 6 memory, only addresses from the user program and from connected nonprivileged subsystems are relocated. If special circumstances dictate that relocated addresses are also required for privileged subsystems, the memory segment must be identified as a data space by entering “ALT” in the ASEL field.

Handling substructures

Substructures are permitted in control blocks that were defined in a high-level language. DAMP allows substructures of this sort to be “revealed” or “hidden”.

To start with, the substructures are ignored when a control block is output, i.e. the fields concerned are output as an “ARRAY OF BYTES”. The names of the fields containing the substructures are prefixed by an asterisk (*).
If you overwrite this character with a “+”, the corresponding substructure is “revealed” and if you overwrite it with a “-”, the substructure is “hidden”.

The choice of transmission key (DUE1 or DUE2) determines whether the substructure is revealed in “edited” format or “compressed” format.

– DUE1 “edited” format. Only one data field of the substructure is displayed below the superordinate name in each line. The field names are indented.

– DUE2 “compressed” format. The substructure is displayed in the same way as a normal control block. It is merely inserted in the appropriate position in the window.

```
DAMP <version> USERDUMP(<ver>) from BS2000(<ver>)   <date> <time>


ESMFHDR                          +00000=0125AB64 TID=000400E4          W4,CBM,L19
000*IF_ID     : 00580201                | 004*RETURNCODE: 0E400009
```

Figure 32: Hidden sample substructure

```
DAMP <version> USERDUMP(<ver>) from BS2000(<ver>)   <date> <time>


ESMFHDR                          +00000=0125AB64 TID=000400E4          W5,CBM,L19
000+IF_ID     :
  000 UNIT    : 0058     =    88
  002 FUNCTION: 02
  003 VERSION : 01
004+RETURNCODE:
  004+STRUCTURED_RC:
    004+SUBCODE         :
      004 SUBCODE2: 0E
      005 SUBCODE1: 40
    006 MAINCODE        : 0009
```

Figure 33: Fully revealed substructure in edited format

```
DAMP <version> USERDUMP(<ver>) from BS2000(<ver>)   <date> <time>


ESMFHDR                          +00000=0125AB64 TID=000400E4          W7,CBM,L19
000+UNIT    : 0058     =    88          | 002 FUNCTION: 02
003 VERSION : 01                       | 004+SUBCODE2: 0E
005 SUBCODE1: 40                       | 006 MAINCODE        : 0009
```

Figure 34: Revealed substructure in compressed format

5.3.2.9 Output in Assembler format

A memory segment displayed in a dump window can be disassembled and displayed in Assembler format. Data sequences which can be interpreted beyond doubt as not being instructions are displayed in the form of DC constants. Output always starts at a half-word boundary. If parts of the disassembled data area are to be displayed as DC constants, the corresponding lines must be marked in the operation code column. When the DUE key is pressed, disassembly of these instructions is disabled, and the corresponding memory locations are displayed as constants.

The Assembler format is determined by the “ASS”, “CAS” or “XAS” entry in the “Output format” field.
If “ASS” is entered, DAMP automatically determines via the CSECT attribute whether /390 or x86 instructions are to be output. You can, however, also determine the format of the disassembly yourself (e.g. if the CSECT attributes are missing or corrupt). /390 instructions are displayed for the “CAS” entry or Xx86 instructions for “XAS”.
In the output, DAMP returns either “ASS” (/390 format) or “XAS” (x86 format) in the “Output format” field.
Address levels and operation codes can be marked in the output, but operation codes only in the case of /390 objects.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


ASTRACE                          +0004C=C0ADE70C SYS=00010001          W4,XAS,L19
C0ADE70C (0004C):448B5C2448          = mov         r11d,DWORD PTR [rsp+X'48'(R1)]
C0ADE711 (00051):90                  = nop
C0ADE712 (00052):458D5570            = lea         r10d,[R13+X'70']
C0ADE716 (00056):41C60226            = mov         BYTE PTR [r10],X'26'
C0ADE71A (0005A):41C6420101          = mov         BYTE PTR [r10+X'01'],X'01'
C0ADE71F (0005F):418BF3              = mov         esi,r11d
C0ADE722 (00062):0FCE                = bswap       esi
C0ADE724 (00064):41897204            = mov         DWORD PTR [r10+X'04'],esi
C0ADE728 (00068):44895C2420          = mov         DWORD PTR [rsp+X'20'(R11)],r11
                                    d
C0ADE72D (0006D):41BF24000000        = mov         R15d,X'00000024'
C0ADE733 (00073):6641C1CF08          = ror         R15w,X'08'
C0ADE738 (00078):6645897A02          = mov         WORD PTR [r10+X'02'],R15w
C0ADE73D (0007D):4489542448          = mov         DWORD PTR [rsp+X'48'(R1)],r10d
C0ADE742 (00082):41BF8C027FC0        = mov         R15d,X'C07F028C'
C0ADE748 (00088):4C0BBC2440010000    = or          R15,QWORD PTR [rsp+X'00000140'
                                    ]
C0ADE750 (00090):448D3503000000      = lea         R14d,[X'C0ADE75A']
CMD:
Key: 1=Help 2=Tsk 3=PCB 4=SC0ADE 5=Dump   6=Dump   7=Dump   8=Dump   9=Dump
```

Figure 35: Example of disassembled output of x86 code

The USE-REGISTER statement (see page 222) can be used to control the symbolic display of the instruction addresses in /390 code but not in X86 code. In this case, the address instructions are no longer displayed as a register and a displacement but as a module-relative address or in the form of a field name in the specified DSECT.

The /390 instruction format used for disassembly is selected automatically by DAMP on the basis of the CPU series used. The MODIFY-OBJECT-ASSUMPTIONS statement can be used to change this preset value. Instruction formats 1 to 5 are available (see the description of the MODIFY-OBJECT-ASSUMPTIONS statement on page 194).

Any USE-REGISTER statements used always apply to all disassembly operations for the same module, even if this is displayed in several dump windows. The DROP-REGISTER statement (see page 189) can be used to cancel these assignments.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


ASTRACE                          +0047E=7151563E SYS=00010001          W5,ASS,L19
7151563E (047E):   D2 03 9008 B010   =   MVC   8(4,R9),WORD004
71515644 (0484):   D2 03 9014 B018   =   MVC   20(4,R9),WORD006
7151564A (048A):   D5 03 B01C A098   =   CLC   WORD007(4),LOC#04F0
71515650 (0490):   47 70 A05C        =   BNZ   LOC#04B4
71515654 (0494):   58 40 A09C        =   L     R4,LOC#04F4
71515658 (0498):   95 01 4A28        =   CLI   2600(R4),1
7151565C (049C):   47 70 A052        =   BNZ   LOC#04AA
71515660 (04A0):   D2 03 9018 A0A8   =   MVC   24(4,R9),LOC#0500
71515666 (04A6):   47 F0 A058        =   B     LOC#04B0
7151566A (04AA):   D2 03 9018 A0AC   =   MVC   24(4,R9),LOC#0504
71515670 (04B0):   47 F0 A062        =   B     LOC#04BA
71515674 (04B4):   D2 03 9018 B01C   =   MVC   24(4,R9),WORD007
```

Figure 36: Output in Assembler format; the following statements were entered:
USE-REG MOD-NA=ASTRACE,REG=10,FOR=*MOD-BASE(DISPL=X'458')
USE-REG MOD-NA=ASTRACE,REG=11,FOR=*CONTR-BLOCK(NAME=WORDLIST)

The statement ADD-LIST-OBJECTS WINDOW=<w> can be used to have the disassembled memory segment output to a list.

5.3.2.10 Output in areas with real addresses

Areas with real addresses are contained in SLED and SNAP dumps as well as in complete VM2000 SLED files after selection of a virtual machine.

If the start address of a memory segment is available as a real address, this segment can be output directly to a dump window without address conversion. This is done by entering the keyword `RM` in the “ASEL” input field in the header line of a dump window and the desired real address in the “Absolute address” input field.

DAMP automatically sets “0” as the 4GB segment if the “ASID” field is not filled. If a real address above 4GB is to be output, the associated 4GB segment must be entered as “ASID” and the relative displacement of the address to the start of the segment must be entered as the “Absolute address”.

The result is displayed in the currently selected output format. However, this can be changed to one of the other output formats either beforehand or later.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


                                +00000=  001000 RM =00000000          W8,D  ,L19
001000 (0000) :  D7C9C401  00000000  00000000  C2E2F2F0   <==>  PID?????????BS20
001010 (0010) :  F0F04040  E5F1F94B  F0C1F0F0  C7F10000   <==>  00  V19.0A00G1??
001020 (0020) :  00000000  E7E5E3F4  FFFFFFFF  72AFA008   <==>  ????XVT4~~~~????
001030 (0030) :  7DD2C040  710011BC  FFFFFFFF  71001264   <==>  'K? ???\~~~~????
001040 (0040) :  713CE380  710011A4  7FA8BCF0  72C6A0E0   <==>  ??T????u"y\0?F??
001050 (0050) :  710011F8  71001648  FFFFFFFF  710011F8   <==>  ???8????~~~~???8
001060 (0060) :  FFFFFFFF  FFFFFFFF  FFFFFFFF  7FD5E000   <==>  ~~~~~~~~~~~~"N??
001070 (0070) :  71001160  7FAE23A8  FFFFFFFF  71294D40   <==>  ???-"??y~~~~??(
001080 (0080) :  FFFFFFFF  FFFFFFFF  FFFFFFFF  FFFFFFFF   <==>  ~~~~~~~~~~~~~~~~
001090 (0090) :  7D6E0A10  FFFFFFFF  7FD3C940  FFFFFFFF   <==>  '>??~~~~"LI ~~~~
0010A0 (00A0) :  7F295710  71001148  7FD1B000  FFFFFFFF   <==>  "???????"J??~~~~
0010B0 (00B0) :  7100114C  71001154  FFFFFFFF  FFFFFFFF   <==>  ???<????~~~~~~~~
0010C0 (00C0) :  FFFFFFFF  7FD3EF80  FFFFFFFF  FFFFFFFF   <==>  ~~~~"L??~~~~~~~~
0010D0 (00D0) :  FFFFFFFF  71F8F8E0  7F302000  7100161C   <==>  ~~~~?88?"???????
0010E0 (00E0) :  FFFFFFFF  FFFFFFFF  FFFFFFFF  71001278   <==>  ~~~~~~~~~~~~????
0010F0 (00F0) :  72925000  7289DC60  72899CC0  7DC20A60   <==>  ?k&??i?-?i??'B?-
001100 (0100) :  733E4D90  FFFFFFFF  FFFFFFFF  FFFFFFFF   <==>  ??(?~~~~~~~~~~~~
001110 (0110) :  710014E0  FFFFFFFF  00000000  00000000   <==>  ????~~~~????????
CMD:
Key: 1=Help 2=Tsk 3=PCB 4=S714CB 5=S71515 6=Dump   7=Dump   8=R-0001 9=Dump
```

Figure 37: Output of area starting at real address 1000; display in the key line: R-0001

Output with real addressing is useful, for example, for the analysis of CCW chains in the memory management tables, for fixed hardware areas (e.g. save areas) or for diagnosis in the case of “overwriters”.

With standard list editing, output with real addresses applies only to the hardware areas provided as defaults. If required, other areas can be output to a list by specifying `ADD-LIST-OBJECTS`.

5.3.2.11 Output in areas with absolute addresses

Absolute addresses occur in a complete VM2000 SLED file. The entire VM2000 system can be addressed using absolute addresses.

An area with absolute addresses can be output in a dump window. This is done by entering the keyword ABS in the “ASEL” input field in the header line of a dump window and the required address in the “Absolute address” input field.

For an absolute address above 4GB, enter the associated 4GB segment in the “ASID” field and the relative displacement of the address to the start of the segment in the “Absolute address” field.
The information is output in the set output format. A different output format can be set before or after this occurs.

```
DAMP <version> SLED(<ver>) from VM2000(<ver>)


                                +00000=  001000 ABS=00000000          W4,D  ,L19
001000 (0000) :  D7C9C401  00001024  00000000  E5D4F2F0   <==>  PID????????VM20
001010 (0010) :  F0F04040  E5F1F14B  F0C1F1F0  F0F00000   <==>  00  V11.0A1000??
001020 (0020) :  00000000  00470101  FFFFFFFF  D7C9C401   <==>  ????????~~~~PID?
001030 (0030) :  00000000  00000000  E2E8E2E2  E3C1D9E3   <==>  ????????SYSSTART
001040 (0040) :  E5F1F94B  F0C1F0F0  C7F10601  00000000   <==>  V19.0A00G1??????
001050 (0050) :  00000000  00000010  E85CE3C5  E2E3C1D4   <==>  ????????Y*TESTAM
001060 (0060) :  C5D5E35C  00000000  00E00000  00E8C4D6   <==>  ENT*?????????YDO
001070 (0070) :  D4C1C9D5  00E00000  0A000000  01E8C4D6   <==>  MAIN?????????YDO
001080 (0080) :  D4C1C9D5  0AE00000  0A000000  02D5C4D6   <==>  MAIN?????????NDO
001090 (0090) :  D4C1C9D5  14E00000  23000000  03D5C4D6   <==>  MAIN?????????NDO
0010A0 (00A0) :  D4C1C9D5  80037E00  800AF000  04D5C4D6   <==>  MAIN??=???0??NDO
0010B0 (00B0) :  D4C1C9D5  00000000  00000000  D5E8C4D6   <==>  MAIN????????NYDO
0010C0 (00C0) :  D4C1C9D5  00000000  00000000  D5E8C4D6   <==>  MAIN????????NYDO
0010D0 (00D0) :  D4C1C9D5  00000000  00000000  D5E8C4D6   <==>  MAIN????????NYDO
0010E0 (00E0) :  D4C1C9D5  00000000  00000000  D5E8C4D6   <==>  MAIN????????NYDO
0010F0 (00F0) :  D4C1C9D5  00000000  00000000  D5E8C4D6   <==>  MAIN????????NYDO
001100 (0100) :  D4C1C9D5  00000000  00000000  D5E8C4D6   <==>  MAIN????????NYDO
001110 (0110) :  D4C1C9D5  00000000  00000000  D5E8C4D6   <==>  MAIN????????NYDO
CMD:
Key: 1=Help 2=Inf 3=PCB 4=A-0001 5=Dump   6=Dump   7=Dump   8=Dump   9=Dump
```

Figure 38: Output as of absolute address 1000

5.3.2.12 Output of dump file sections

The structure of dump file sections is determined by the dump generator (SLED, SNAP or CDUMP).

Parts of dump file sections can be viewed in a dump window. You do this as follows: In the header line, type the keyword “SCT” into the “ASEL” input field and the name of the section you want into the “ASID” input field.

The output is displayed in the currently selected format. You can switch to other output formats either beforehand or subsequently.

DAMP supports the output of dump file sections with a sequential, homogeneous, or mixed structure; it does not support the output of dump file sections with an inhomogeneous structure.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


                                +00000=  001000 SCT=SLEDMEM           W7,D  ,L19
001000 (0000) :  10480100  01000000  00000000  C1C4E2F1   <==>              ADS1
001010 (0010) :  010ED000  010ED400  010EDC00  D4E2C7C2   <==>        M     MSGB
001020 (0020) :  E4C6C67A  01111000  E2C4E37A  01145000   <==>  UFF:    SDT:  &
001030 (0030) :  0116D500  0116D600  00000000  00000000   <==>    N   0
001040 (0040) :  00000000  00000000  00000000  00000000   <==>
001050 (0050) :  00000000  00000000  00000000  00000000   <==>
001060 (0060) :  00000000  00000000  00000000  00000000   <==>
001070 (0070) :  00000000  00000000  40D9C9C7  C8E3E240   <==>          RIGHTS
001080 (0080) :  D9C5E2C5  D9E5C5C4  D7D7D7D7  D7D7D7D7   <==>  RESERVEDPPPPPPPP
001090 (0090) :  010010A0  010013F0  00350000  00000000   <==>         0
0010A0 (00A0) :  01001000  D5E2C9C9  D7D3C8E6  F000F000   <==>      NSIIPLHW0 0
0010B0 (00B0) :  01010000  C5E3E2E5  D7404040  F002F700   <==>      ETSVP   0 7
0010C0 (00C0) :  0103F700  D5E2C9C4  C5D94040  F0006400   <==>    7 NSIDER  0
0010D0 (00D0) :  01045B00  D5E2C9C5  E7D4C7E3  F0003000   <==>    $ NSIEXMGT0
0010E0 (00E0) :  01048B00  D5E2C9C9  D7D3C3D6  F000A800   <==>      NSIIPLC00 y
0010F0 (00F0) :  01053300  D5E2C9C9  D7D3C4D4  F000F800   <==>      NSIIPLDM0 8
001100 (0100) :  01062B00  D5E2C9C9  D7D3C4E3  F0004000   <==>      NSIIPLDT0
001110 (0110) :  01066B00  D5E2C9C9  D7D3C9C8  F0006400   <==>    , NSIIPLIH0
CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=s-0001 8=Dump   9=Dump
```

Figure 39: Output of the SLEDMEM dump file section; s-0001 is shown in the key line

5.3.2.13 Tracing chains

If you have found the beginning of a longer chain in a dump window and want to trace that chain, the best method of doing this is as follows:

– Keep the beginning of the chain in one dump window (e.g. W4) and assign two free dump windows or windows which are no longer needed (e.g. W6 and W8) by means of the statement

```
MODIFY-SCREEN-LAYOUT FIRST=6(SIZE=5), SECOND=8(SIZE=5), THIRD=4
```

– Mark the first chaining address in W4 with + DUE (the third window on the screen). The marked address appears in window W6.

– Alternately mark the next chaining address in the two upper windows W6 and W8 until the desired address is reached.

This method has the advantage that tracing can be restarted at any time from the previous address or even the first address if you make a mistake.

If this safety precaution is not desired, or if only a certain memory segment in the chaining sequence is needed for further processing, the following method can be used:

– Assign the entire screen to the window in which the first chaining address is displayed (e.g. W4) by means of the statement

```
MODIFY-SCREEN-LAYOUT FIRST=4(SIZE=19)
```

– Mark the chaining address in this (single) window W4 until the desired memory segment is found.

This method ensures that a window whose old contents are needed for further processing is not inadvertently overwritten with the chaining addresses.

Another useful feature is the fact that any DSECT selected for a window remains stored. Consequently, if various elements of the chain are displayed sequentially in the same window, each area is immediately edited in the symbolic format of the selected DSECT.

5.3.2.14 Output of system trace tables (special window: TRACE)

The SHOW-EDITED-INFORMATION statement allows you to output the system trace table in edited format in a specific window.

```
SHOW-EDITED-INFORMATION INFORMATION=*TRACE-TABLE-EDIT, WINDOW=<w>
```

Following the call, the trace table entries of all tasks contained in the diagnosis object are displayed. The contents of the “Task select” input field can be changed to a <tid> or to ALL. The number of a logical machine or the value ALL can be entered in the “LM” field.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


SEQ# GP LM =ALL TID PS P_COUNTER      IDENTIFIER     (TID=ALL     )   W9,TRC,L19
   1 00 01          SI ETMSLMG +00058 CRSH NRTC516
  ?2 00 01          SI ETMSLMG +00058 PROG 58 Inv Opcd PCB=00000000 R13=00000000
   3 00 01          WT SYSTEM IDLE    I/O  0000   00C3 CSW=00929008 CSW=0C000000
   4 00 01 00010002 TP*ETMBON1 +001D6 SVC  FA $BOWT    LOC=F1BE43B8 BRS=7125C8C0
?  5 00 01 0001000E TP DQPAM   +01764 SVC  D7 $XCPW    PCB=71175918  R1=733C5028
?  6 00 01 00010008 TP*ETMBON1 +001D6 SVC  FA $BOWT    LOC=F1BEE230 BRS=7125C880
   7 00 01 0001000C TP*ETMBON1 +001D6 SVC  FA $BOWT    LOC=F12A86CE BRS=7125C740
   8 00 01 0001000C TP NBCCNTS +001F0 SVC  D5 $EXCP    PCB=71180720  R1=71FF9F00
   9 00 01 00010008 TP*ETMBON1 +001D6 SVC  FA $BOWT    LOC=F1BEE230 BRS=7125C880
? 10 00 01 000100C1 TP*ETMBON1 +001D6 SVC  FA $BOWT    LOC=FC688E82 BRS=73895B00
  11 00 01 000100C1 TP NEHMSTAT+0007E SVC  EA $FNDT    PCB=73286008  R1=70FFAA10
  12 00 01 00010075 TP*ETMBON3 +001EA SVC  FA $BOWT    LOC=FF512072 BRS=7125D440
? 13 00 01 00010075 TP EMMREQM1+004E0 SVC  F6 $UNMASK  LOC=F12188CC ATT=80EF0000
  14 00 01 00010075 TP DISTRIB +00F72 SVC  EA $FNDT    PCB=732861D8  R1=6EC83020

CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=TRAC   9=OPTS
```

Figure 40: Output format of the system trace table

Besides selecting a task, you can also modify the length of a window and page within a window. See “Paging in a diagnostic window” on page 86 for more details.

The entries in “SEQ#” as well as all the fields that contain the addresses can be marked. You mark the address of a system trace table entry with the sequence number. This is useful if you want to look at an entire entry.

The CPU number and the logical machine number are shown in the GP and the LM column of the trace list.

By default, DAMP sorts the trace entries chronologically on the basis of the time stamp, which is also recorded. If required, however, output can be specified for a specific logical machine. This is done by entering a number in the LM field in the header line. Note that it is not possible to select a task at the same time as a logical machine and vice versa.

5.3.2.15 Output of memory attributes (special window: MEMATTR)

Edited memory attributes are displayed by entering the following statement:

```
SHOW-EDITED-INFORMATION INFORMATION=*MEMORY-ATTRIBUTES, WINDOW=<w>
```

This special window is used to display the table containing the attributes of the allocated virtual memory pages. It can also be used to have the memory class limits displayed.

Information can be output on a specific user address space or, if so desired, for the system address space. The attributes can be shown in symbolic format or in the original output format of the VAT (Virtual Attribute Table).

In order to enable these options, the title line of the window contains the two fields “ASN” and “Output format”.
A TID or the string “SYSTEM” can be entered in the “ASN” field allowing you to specify the address space.
The following abbreviations can be entered in the “Output format” field:
– “LIM” for the memory class limits
– “SYM” for symbolic representation
– “HEX” for hexadecimal representation of the first byte of the virtual attribute
– “HX2” for hexadecimal representation of the second byte of the virtual attribute

The output of the page attributes contains one line for every 16 pages, and this line, in turn, contains a three-character entry for each page. Depending on the output format selected, these entries contain either a symbolic representation of the attributes of this page (memory class, partial page indicator, privilege indicator) or the original byte from the VAT in hexadecimal format.

Ellipses (...) indicate that this page is not allocated. If the virtual attributes cannot be accessed (normal for area dumps, but also possible for other dump types), (...) only indicates that the page is not contained in the dump file.

A question mark and two dots (?..) indicate that the page is contained in the dump file but that the virtual attributes cannot be accessed.

An asterisk (*) in the first column of a line indicates that the entries of the preceding page have not changed since the last time an entry was output. The attributes for this page are displayed in the extreme right of the preceding line.

Memory pool segments are indicated by the string “Pool” in the right margin.

The significance of the symbols for the SYM output or of the bits for the HEX and HX2 outputs is explained in the header line.

When displaying page attributes, a page number may be entered in the first line in the “PAGE” field. The edited output then begins at the specified page. Page numbers (up to 8 hexadecimal digits) are within the range of X'0' to X'1FE00000'.

```
DAMP <version> SYSDUMP(<ver>) from BS2000(<ver>)    <date> <time>


The Symbols mean: <Class>,(P)artial-Pg,(N)onpriv-Pg      ASN=SYSTEM   W9,SYM,L19
 PAGE  :  0   1   2   3   4   5   6   7   8   9   A   B   C   D   E   F  Remarks
 00C0X : ShrBase  : 00C00000
 00C0X : ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
*00F8X : ... ... ... ... ... ... ... ... ... ... ... ... 4P- 4P- ... 4--
*00FFX : 4-- 4-- 4-- 4-- 4-- 4-- 4-- 4-- 4-- 4-- 4-- 4-- 4-- 4-- ... ...
 7100X : Cl1Base  : 71000000
 7100X : 1-- 1-- 1-- 1-- 1-- 1-- 1-- 1-- 1-- 1-- 1-- 1-- 1-- 1-- 1-- 1--
*713FX : 1-- 1--
 713FX : Cl2Base  : 713F2000
 713FX :         2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2--
*71E0X : 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2-- 2--
 71E1X : Cl2Limit : 71E0FFFF
 71E1X : ... ... ... ... ... ... ... ... ... ... ... ... ... ... ... ...
*71E6X : ... 3-- 3-- 3-- 3P- 3P- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3--
*71ECX : 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3P- 3P- 3-- 3-- 3-- 3-- 3-- 3--
*71EEX : 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3P- 3P- 3-- 3--
*71FBX : 3P- 3P- 3P- 3P- 3P- 3P- 3P- 3P- 3P- 3P- 3P- 3P- 3P- 3P- 3P- 3P-
 71FCX : 3P- 3P- 3P- 3P- 3P- 3P- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3-- 3--
CMD:
Key: 1=Help 2=Plk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=Dump   9=MEMA
```

Figure 41: Output of memory attributes

```
DAMP <version> SYSDUMP(<ver>) from BS2000(<ver>)    <date> <time>


Class Limits for System / User Space :                   ASN=SYSTEM   W9,LIM,L19
 PAGE    Space                Start      End

 00C00 : Shared Code        = 00C00000 - 00FFFFFF

 71000 : Class 1            = 71000000 - 713F1FFF

 713F2 : Class 2            = 713F2000 - 71E0FFFF

 71E10 : Class 3/4          = 71E10000 - 7FFFEFFF
```

Figure 42: Output of memory class limits

5.3.2.16 Output of tables with task-specific values (special window: TABLE)

The TABLE function provides a clear overview of task-specific values for all tasks contained in the diagnosis object.

Following the

```
SHOW-EDITED-INFORMATION INFORMATION=*TASK-TABLES, WINDOW=<w>
```

call, the requested dump window W4 - W9 or W21 - W99 is displayed as a TABLE window in the first position of the current window size, but does not contain any output values in the diagnosis area as yet.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>

     TID dsectfld, dsectfld->dsectfld ...                              W6,TBL,L19
```

Figure 43: Dump window after calling the TABLE function

The user can then enter, in the header line, a list of field names of the task-specific DSECTs ETCB, ETCB, EJCB and EVUMT, separated by blanks. The following output then consists of one line per task with the contents of these fields.

If task-specific data fields from other control blocks are to be shown, the path to the data field required must be described unambiguously. Enter a sequence of field names in the header line of the window in the following form:

field-name1 -> field-name2 -> field-name3 -> ... -> field-namex

The field name field-name1 must come from one of the DSECTs mentioned above, which can be localized automatically, and must point to the beginning of the structure containing field-name2. field-name2, in turn, points to the beginning of the structure containing field-name3, and so on. The required field with the field name field-namex is at the end of the list.

If one of the specified field names contains a value of 0 or if the address indicated has not been allocated, DAMP stops resolving the sequence of field names.

If, for instance, you wish to identify tasks in which more than three files are open, the following construction can be used:

ETCBTFT -> IDMFRLNK -> IDMFRLNK -> IDMFRLNK -> IDMFRLNK

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


     TID ETCBTSN  ETCBTFT->IDMFRLNK->IDMFRLNK->IDMFRLNK->IDMFRLNK     W6,TBL,L19
000100DD    FTCP 70F9A000->70F9A0A8->70F9A150->70F9A1F8->00000000
000100DE    OS91 70F44000->
000900DF    XAE8 00000000->
000900E0    XAEY 00000000->
000900E1    XAE9 00000000->
000D00E2    OTTK 70F9A000->
000900E3    PR03 70F60000->00000000->
000100E4    PR1C 70F60000->00000000->
000D00E5    OTTB 70F9A000->70F9A0A8->70F9A150->70F9A1F8->70F9A2A0
000100E6    XAD7 00000000->
000D00E7    OTTQ 70F9A000->70F9A0A8->70F9A150->70F9A1F8->70F9A2A0
000400E8    OTR6 00000000->
000E00E9    OTTT 70F9A000->
000A00EA    OTRM 00000000->
000200EB    OTVE 70F9A000->70F9A0A8->70F9A150->70F9A1F8->70F9A2A0
000200EC    OTVF 70F9A000->70F9A0A8->70F9A150->70F9A1F8->70F9A2A0
000900ED    OTRT 00000000->
000200EE    OTT8 70F9A000->70F9A0A8->70F9A150->70F9A1F8->70F9A2A0
CMD:
Key: 1=Help 2=Plk 3=PCB 4=ETCB   5=ESTK   6=Dump   7=Dump   8=TABL   9=Dump
```

Figure 44: Output of chaining sequences; specification in input line:
ETCBTSN(C) ETCBTFT->IDMFRLNK->IDMFRLNK->IDMFRLNK->IDMFRLNK
The TSN is also output in printable format

The output formats of the fields match their definitions in the DSECT; if desired, however, the format can be modified by explicitly specifying a format character in parentheses after the field name. The permitted format characters are:

C Display in character format

X Display in hexadecimal format

I Display as an integer

You can page through the output by entering F3 /+, ++, F1 /- , --, +n and -n; see “Paging in a diagnostic window” on page 86 for further details.

5.3.2.17 Output of information on subsystems (special window: SUSY)

The SUSY function enables the DAMP user to display information on

– the BS2000 nucleus (Control Program, CP)
– all subsystems that were loaded with DSSM
– all loaded user program contexts

The Control Program and the user program contexts are assigned pseudo subsystem names by DAMP. The Control Program receives the subsystem name 'CP', and all user contexts are given the subsystem name 'USERPROG'. If there are several loaded user contexts in a task, the uniqueness of the subsystem name is ensured by DAMP by means of an internally assigned 'USERPROG' version number CTXNRxxx (where 'xxx' is a sequential number starting with 001, 002, ...).

User dumps contain information on the following subsystems:
– all loaded user contexts
– all nonprivileged subsystems connected to the task

This information is usually missing in area dumps.

In the case of system dumps, SLEDs, SNAPs and the active system, details on the following subsystems are displayed:
– Control Program
– all loaded privileged and nonprivileged subsystems

SUSY is called as follows:

```
SHOW-EDITED-INFORMATION INFORMATION=*SUBSYSTEM-INFORMATION, WINDOW=<w>
```

For information on the layout of the generated window and an explanation of the fields, see page 116.

You can page within the SUSY window with F3 /+, ++, F1 / -, --, +n and -n; see “Paging in a diagnostic window” on page 86 for further details.

ENTRY names are not supported in the SUSY window.

The header lines of the SUSY window

The header lines of the SUSY window contains several input fields in which the subsystems, holder tasks or CSECTs to be output can be entered.

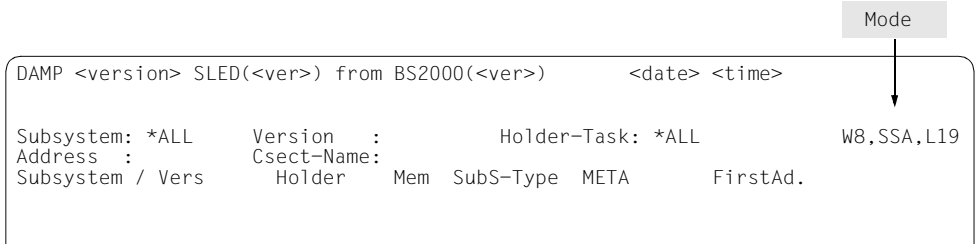


Figure 45: Header lines of the SUSY window

Possible entries in the header lines of the SUSY window

It is possible to combine new entries in the various fields with previous entries, i.e. entries made in the input fields earlier which are still being displayed retain their validity and new entries can be made.

– **Subsystem** field

  If no entry is made in any other field, a list of the CSECTs of this subsystem is displayed. Entering *USER is equivalent to entering USERPROG.
  If, however, a CSECT name or an address is also entered, the system only searches the specified subsystem.
  If Subsystem=*ALL and, for example, a CSECT name is entered, all the subsystems are searched for this CSECT.
  If only Subsystem=*ALL is specified, the system switches to the subsystem list. Any entry made in the “Holder-Task” field is taken into account to only a limited extent.
  If you mark a subsystem name in one of the output lines, the corresponding CSECT list is displayed.

  The context names are output in the overview window as an alternative to the other information on the subsystems. To do this, you must either mark a subsystem version or enter “CTX” in the mode field.

– **Version** field

  The version of the desired subsystem can be entered in this field.

– **Holder-Task** field

  If a TID is entered in the field “Holder-Task”, the list of subsystems is abbreviated to those whose holder task matches the specified TID. The same effect can be achieved by marking the TID under “Holder” in one of the output lines.

At the same time, the system attempts to change the active task (the task with the specified TID is activated if necessary). One side-effect of this can be that a user program is added to or disappears from the end of the subsystem list.
The original overview can be redisplayed by entering *ALL in this field or SSA in the “Mode” field.

– **Csect-Name** field

If a CSECT name is entered in the field, the output format changes and a list of all subsystems in which this CSECT occurs is displayed.
You can restore the subsystem list by entering *ALL in the “Subsystem” field.

This function is also permitted for modules of the CP (Control Program) in order, for example, to display their identification field (ETPND).

– **Address** field

An address which is to be localized, i.e. converted into a module and a displacement, can be entered in the field “Addr”. The resulting output is in the format of the CSECT list and contains all modules within which this address could lie.

– **Mode** field

This field is a combined input/output field and has the following meaning:

SSA	A list of all subsystems is displayed.
SSC	Only the subsystems connected with the current task are displayed.
SSH	A list of all subsystems whose holder task matches the specified TID is displayed (only as an output field).
CTX	The context name is output for all subsystems.
INF	CTX is reversed.
CS2	The layout of the CSECT list is switched. Instead of the ETPDN, the P mode and the HSI byte is output for each CSECT. This mode is only supported for diagnostic objects of x86 servers.
CS1 or CSE	CS2 is reversed.
EDT	The data corresponding to the settings of the SUSY window is output in its entire length to the current EDT area.
LST	The entire subsystem and CSECT information of the object is output to the current EDT area.

The current task (in the SLED or SNAP) is the last task selected by the person performing the diagnosis (by marking an item in window W2 or by explicitly entering a TID or TSN etc.).

Layout of the subsystem list

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


Subsystem: *ALL     Version   :          Holder-Task: *ALL            W8,SSA,L19
Address  :          Csect-Name:
Subsystem / Vers      Holder    Mem  SubS-Type  META       FirstAd.
CP        / <ver>     00000000  SYS  Nuc                   71000000
AID       / <ver>     0001001D  SYS  Priv       7CFDF000   7CC00000
AIDSYS    / <ver>     0001001D  SYS  Priv       7D45D000   7D3F5000
AIDSYSA   / <ver>     0001001D  SYS  Priv       7EF1A000   73199000
ANITA     / <ver>     0001001D  SYS  Priv       7C7FA000   7D0A7000
ASE       / <ver>     0001001D  SYS  Priv       7D5EC000   7D508000
ASTI      / <ver>     0001001D  SYS  Priv       7E5CA000   7E92C000
BCAM      / <ver>     0001001D  SYS  Priv       7F436000   72BE8000
BLSSERV   / <ver>     0001001D  SYS  Priv       7F50D000   7F533000
BLSSYS    / <ver>     0001001D  SYS  Priv       7FACB000   7FABC000
CALENDAR  / <ver>     0001001D  SYS  Priv       7FAE8000   7F82B000
CAPRI     / <ver>     0001001D  SYS  Priv       7E1A8000   7DA3C000
CCOPY     / <ver>     0001001D  SYS  Priv       7D7FE000   7D995000
CMX-TP    / <ver>     0001001D  SYS  Priv       7E1A2000   7D985000
CPR       / <ver>     0001001D  SYS  Priv       7E1E1000   7D96C000
CRYPT     / <ver>     0001001D  SYS  Priv       7E5C0000   732AF000
CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=SUSY   9=Dump
```

Figure 46: Subsystems overview

The individual columns of the subsystem list shown in figure 46 have the following meanings:

– The column **Subsystem**
  Name of the subsystem.
  If a field in this column is marked, output switches to the CSECT list format for this subsystem.

– The column **Vers**
  Version of the subsystem.
  If a field in this column is marked, a switch occurs to output of the context name for the marked subsystem.

– The column **Holder**
  TID of the holder task (if it exists).
  If a field in this column is marked, an abbreviated subsystem list with all subsystems “held” by this task is displayed.

– The column **Mem**
  Memory area in which the subsystem was loaded. The value “SYS” for system memory or “USR” for user memory is displayed.

– The column **SubS-Type**
  Subsystem type.
  This column can contain the values Nuc, Priv, NPriv, TU, Usr-Ctx, Pool-Ctx, TaskLoc, and undefined.

– The column **META**
  Address of the ANITA metadata.
  If a field in this column is marked and one of the keys P4 to P9 is pressed, the ANITA metadata for this subsystem is displayed in the desired window. This area contains a list of load information that was generated for the subsystem.
  This field is empty for the subsystem CP.
  There is likewise no display for subsystems loaded before DSSM.

– The column **First Ad.**
  Start address of the subsystem.
  This field contains the lowest address of the found CSECTs of the subsystem. If no load information for the subsystem is available, the start address is not supplied.

Layout of the CSECT list

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


Subsystem: AIDSYSA  Version   : <ver>     Holder-Task: 0001001D
W8,CSE,L19
Address  :          Csect-Name:
Subsystem/Version Address  = Module   + Reladd  Length    ETPND-Info
                  73199000 = ASASHS   +      0  00001E00  ASASHS   897A 20080730
                  7EEF0000 = ASAIDENT +      0  00000600  ASAIDENT 866A 20080730
                  7EEF0600 = ASYTPRV  +      0  00000EC0  ASYTPRV  810A 20080730
                  7EEF14C0 = ASAFNAT  +      0  00000980  ASAFNAT  890A 20080730
                  7EEF1E40 = ASACMD   +      0  00000980  ASACMD   8901 20080730
                  7EEF27C0 = ASACMD@@ +      0  00001B40  ASACMD@@ 8901 20080730
                  7EEF4300 = ASASSD   +      0  00000080  ASASSD   8901 20080730
                  7EEF4380 = ASAENAT  +      0  00000240  ASAENAT  8991 20080730
                  7EEF45C0 = ASAENAT@ +      0  000017C0  ASAENAT@ 8991 20080730
                  7EEF5D80 = ASAEVT   +      0  000002C0  ASAEVT   8911 20080730
                  7EEF6040 = ASAEVT@@ +      0  00002A00  ASAEVT@@ 8911 20080730
                  7EEF8A40 = ASAUTIL  +      0  00000900  ASAUTIL  8971 20080730
                  7EEF9340 = ASAUTIL@ +      0  00001900  ASAUTIL@ 8971 20080730
                  7EEFAC40 = ASASHC   +      0  00000900  ASASHC   8971 20080730
                  7EEFB540 = ASASHC@@ +      0  0000AE80  ASASHC@@ 8971 20080730
                  7EF063C0 = ASASH2   +      0  00000880  ASASH2   8971 20080730
CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=SUSY   9=Dump
```

Figure 47: List of CSECTs of the subsystem AIDSYSA

The columns of the CSECT list shown in figure 47 have the following contents:

– The **Subsystem** column
shows the name of the subsystem only if CSECTs which belong to several subsystems are displayed. Otherwise, this field is empty, and the subsystem name displayed in the title line applies.

– The **Address** column
contains the address of the CSECT or an address within the CSECT. The former is the case only if the list was generated by entering an address in “Address” field. This field can be marked. If, for example, one of the keys P4 to P9 is then pressed, the selected window is positioned to the marked address.

– The **Module** and **Reladd** columns
show the address from the “Address” column after conversion to module-relative form.

– The **Length** column
contains the module length.

– The **ETPND-Info** column
contains the ETPND information with the module name, version number and the compilation date.

In the case of prelinked modules, the module length may be zero. In system, user or area dumps, the ETPND information may be missing if the associated virtual page is not contained in the diagnosis object, since the dump generator only stores the referenced code pages.

In the standard window, CSECTs of all subsystems can be specified when localizing memory segments. Even the automatic relocation of address fields considers the CSECTs of all subsystems. This means that SUSY is no longer required for localizing CSECTs. However, this applies only if the CSECT names in the scope covered by all the subsystems are unique. If this condition is not met (e.g. in the NKVT and NKVD subsystems or if different versions of the same subsystem coexist), this procedure always displays the first matched name. Targeted localization is then only possible using SUSY or the SEARCH-IN-SUBSYSTEM statement (see page 209).

– The **PMODE/HSI** column
contains, for diagnosis objects of x86 servers, information on the processor mode in which the code is executed and on the type of code generation. This output occurs in the CS2 mode.

The different possible types of output have the following meanings:

PMODE	**Meaning**
CISC	emulation in /390 mode
X86	native on x86 server
HSI	**Meaning**
R	mixed binary_no
U	mixed binary_yes
00	BLS information is not available or is obsolete. This value is generally displayed in the case of CISC coding.

You can use the keys and entries normally used for paging in the SUSY overview window. The paging functions +/F3, ++, - / F1, --, +n, -n are supported.

There are some class 5 subsystems which occupy different address space strips in the holder task and in the connected user tasks. The information used by DAMP for localization contains the addresses which are valid in a user task. However, the expected modules will not be found at this location in the holder task.

5.3.2.18 Information on system files and sections of the dump file (special window: FILE)

The FILE function is used for the overview, display, output of lists and generation of system files contained (stored) in dump files.

You also get an overview of all sections of the dump file that are not empty. All system files are stored as sections in the dump file.

If you enter
`SHOW-EDITED-INFORMATION INFORMATION=*DUMPED-SYSTEM-FILE, WINDOW=<w>`
the required dump window W4 - W9 or W21 - W99 is output as the uppermost window in the current window size in the form of an overview screen and select screen in INF mode. All sections contained in the dump file are displayed on this screen. You can page using the usual scroll commands.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


Section   Available Modi    * Overview and Selection Screen *         W8,INF,L19
          Mark mode of wanted section to select  - or enter right above
          "LST" to list this overview, "ALL" for more, "INF" for compact info.
SLEDLOG   DSP  EDT  LST  GEN
          Loggingfile of dump generator
CONSLOG   DSP  EDT  LST  GEN
          Dumped Systemfile  $SYSAUDIT.SYS.CONSLOG.<date>.075.001
EQUISAMQ  GEN
          Dumped Systemfile  $TSOS.EQUISAMQ
HELFILE   GEN
          Dumped Systemfile  $TSOS.SYS.HEL.<date>.065621
MSCFTRAC  GEN
          Dumped Systemfile  $TSOS.SYS.MSCF-TRACE.<date>.166.075.001
REPLOG    DSP  EDT  LST  GEN
          Dumped Systemfile  $SYSAUDIT.SYS.REPLOG.<date>.075.01
SERSLOG   GEN
          Dumped Systemfile  $TSOS.SYS.SERSLOG.<date>.075.01
SJMSFILE  GEN
          Dumped Systemfile  $TSOS.SJMSFILE.WORK
CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=FILE   9=OPTS
```

Figure 48: Layout of the generated overview and selection screen

The following entries are possible in the mode field:

ALL Switch to a detailed overview. It contains an edited form of the saved catalog entries of the system files contained in the overview.

LST Output of the detailed overview in accordance with *SYSLST.

INF Revert to the compact overview mode and select mode, if the ALL mode was previously set.

The edit modes available for this section and a brief description of the section are output directly beside the section name. The relevant section is selected and edited by marking an edit mode.

The following editing modes can be selected:

DSP Display the contents of the section in the diagnostic window.

LST Output the contents of the section in accordance with *SYSLST.

EDT Call EDT and read the contents of the section into an EDT area.

GEN Generate a file from the stored system file.

Note on the generation of system files

DAMP does not perform automatic conversion of the file formats.
For example, for a file with PAMKEYs, a disk that supports this format must be accessible. This information does not apply to the generation of REPLOG, CONSLOG or SLEDLOG.
Using ADD-FILE-LINK and the section name as a link name, data media, names, etc. can be agreed for the generation. If no link name exists, DAMP generates the file under the caller's user ID with a name automatically specified by DAMP.

If a mode was marked for a section, and DSP is also available for this mode, the layout of the window changes to the "Layout for a selected section" after the requested processing has been executed (e.g. on returning from EDT). The following input options are available in the title line of this "Layout for a selected section":

Possible entries in the title line for a selected section

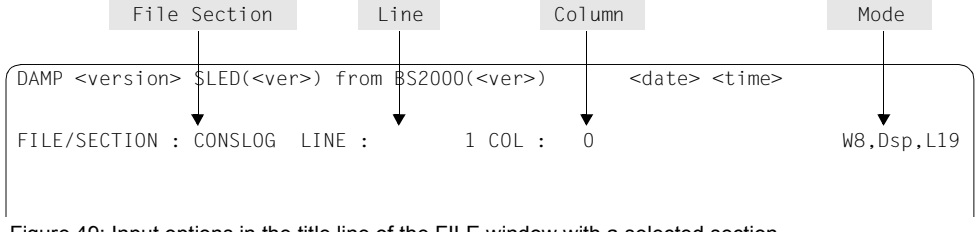


Figure 49: Input options in the title line of the FILE window with a selected section

– **FILE/SECTION** field
Here the name of a system file or a dump section can be specified (see above).

– **Line** field
Here the number of the first line to be displayed can be specified.

– **Column** field
Here the number of the first column to be displayed can be specified.

– **Mode** field
The following six modes exist:

Dsp	Display information in window (default)
Edt	Call EDT and transfer information to the current EDT work area
Gen	Generate as BS2000 file
Lst	Output information to SYSLST
Inf	Revert to the overview and selection screen (INF mode)
All	Revert to the ALL mode of the overview and selection screen

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


FILE/SECTION : REPLOG   LINE :      140 COL :   0                    W8,Dsp,L19
 REP 001F4 103 X'92A106B2A2E1001826F6000892F80016' 0000 FA0434308692 2#1DPSIMPO
 REP 00204 103 X'37180001A2F8001626590180032000 8' 0000 0A0434308692 2#1DPSIMPO
 REP 00214 103 X'00000000'                         0000 FA0434308692 2#1DPSIMPO
 REP 004FC 112 X'24090006000E682591B800003404007A' 26AD FA0434354251 2#1NASDSPP@
 REP 0050C 112 X'1304001500000000091B800003404004B' 25A4 BA0434354251 2#1NASDSPP@
 REP 0051C 112 X'1304001100000000025AD00012529FFFF' 0320 EA0434354251 2#1NASDSPP@
 REP 0052C 112 X'1520FFF5000000000129482610000000A' 2405 3A0434354251 2#1NASDSPP@
 REP 0053C 112 X'00000000'                         25A6 3A0434354251 2#1NASDSPP@
 REP 0458C 114 X'1000'                             1704 EA0434434307 2#1NSPSERV@
 REP 02DC8 142 X'92D8001433040008108000370000000 0' 92D8 DA0434520058 2#1DOCPFA@@
 REP 02DD8 142 X'92D8001034040001170400030000000 0' 92D8 0A0434520058 2#1DOCPFA@@
 REP 02DE8 142 X'1000000C00000000'                 1000 3A0434520058 2#1DOCPFA@@
 REP 02E28 142 X'8EB700801000001E00000000'         2405 DA0434520058 2#1DOCPFA@@
 REP 0303C 142 X'8EB7016C16E0010F00000000'         8EB7 2A0434520058 2#1DOCPFA@@
 REP 03480 142 X'8EB700801000FF0800000000'         0000 EA0434520058 2#1DOCPFA@@
 REP 07B2C 119 X'267901100320000800000000'         8EF6 AA0434806307 2#1DPSCMGR@
 REP 001B0 119 X'8EF6000412C0000400000000265 90F00' 0000 3A0434806307 2#1DPSCMGR
 REP 001C0 119 X'0320000800000000 8E570FE002E0B009' 0000 DA0434806307 2#1DPSCMGR
CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=FILE   9=OPTS
```

Figure 50: Layout of the FILE window after selection of system file REPLOG as of line 140

5.3.2.19 Information on AUDIT tables (special window: AUDIT)

The AUDIT function is used to display the AUDIT tables in a SLED or system dump (hardware and linkage AUDIT).

If you enter
`SHOW-EDITED-INFORMATION INFORMATION=*AUDIT-TABLE-EDIT, WINDOW=<W>`
the required dump window W4 - W9 or W21 - W99 is the first window displayed in the current window length. If the dump contains AUDIT tables, one of them is selected and displayed. The title line of the window contains several input fields, by means of which the various AUDIT table types and areas and the required task can be selected.

In addition to the hardware AUDIT, there is also the linkage AUDIT in processor-local (SIH or SIH+TPR) and task-local (TPR and TU) forms. The AUDIT tables are selected by DAMP after the command `SHOW-EDITED-INFORMATION INFORMATION=AUDIT-TABLE-EDIT` in the sequence “task-local hardware AUDIT (TPR) -> task-local hardware AUDIT (TU) -> task-local linkage AUDIT (TPR) -> task-local linkage AUDIT (TU) -> processor-local linkage AUDIT. The first table found is displayed.

Hardware audit will only be supported on /390 servers.

Possible entries in the title line

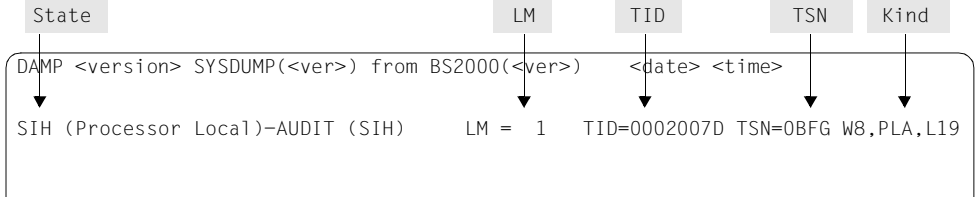


Figure 51: Entries in the title line of the AUDIT window

– **State** field
Here the program status of the required AUDIT tables can be specified:

SIH	System interrupt handling (only for “Kind = PLA”)
TPR	Task privileged (only for “Kind = LKA” or “Kind = HWA”)
TU	Task unprivileged (only for “Kind = LKA” or “Kind = HWA”)

– **LM** field
Indicates the hexadecimal number of the LM (only for “Kind = PLA”)

– **TID** field
Indicates the task identifier (only for “Kind = LKA” or “Kind = HWA”)

– **TSN** field
Indicates the task sequence number (only for “Kind = LKA” or “Kind = HWA”)

– **Kind** field
  AUDIT table type:

  PLA Processor-local linkage AUDIT
  LKA Task-local linkage AUDIT
  HWA Hardware AUDIT

The paging functions+/[F3], ++, - / [F1], +n, -n are supported for AUDIT table output. See “Paging in a diagnostic window” on page 86 for further details.

All addresses can be marked. The addresses are shown in virtual format and where possible edited symbolically (module name + offset).

When the processor-local linkage AUDIT is displayed, the input fields “TID” and “TSN” are not evaluated.

If an AUDIT table is output, the second line shows the addresses of the AUDIT management area (“EXVTLAUD”) and the relevant AUDIT trace table (“AuditTable”). The “AuditTable” field is marked “(current)” if the displayed AUDIT was active at the time of the dump. If the displayed AUDIT was placed in the DISCONTINUE state by the HOLD-LINKAGE-AUDIT or HOLD-HARDWARE-AUDIT command before the dump, “(obsolete)” is added to the “AuditTable” field.

Layout of an AUDIT window

```
DAMP <version> SYSDUMP(<ver>) from BS2000(<ver>)    <date> <time>


SIH (Processor Local)-AUDIT (SIH)     LM =  1   TID=0002007D TSN=0BFG W8,PLA,L19
EXVTLAUD=72408C40   AuditTable (current)=7241E000      ">" indicates LAST branch
> F10053A0 NLCNLMAN+003A0 |  F10450E0 EMMPGSRV+00860 |  F1064AC0 ETMPI$X +00000
  F1062648 EMMPLAT0+00288 |  F1064DE8 ETMPI$X +00328 |  F1064AC0 ETMPI$X +00000
  F1062648 EMMPLAT0+00288 |  F106084C ETMTIM$X+002CC |  F1062648 EMMPLAT0+00288
  F1006E30 NLCNLMAN+01E30 |  F1006CAC NLCNLMAN+01CAC |  F1064F72 ETMPI$X +004B2
  F1056158 ETMPSUBR+01298 |  F1006E30 NLCNLMAN+01E30 |  F1055C40 ETMPSUBR+00D80
  F10559B8 ETMPSUBR+00AF8 |  F1055660 ETMPSUBR+007A0 |  F1060EA0 ETMTIM$X+00920
  F1056028 ETMPSUBR+01168 |  F1056080 ETMPSUBR+011C0 |  F1006CAC NLCNLMAN+01CAC
  F1055790 ETMPSUBR+008D0 |  F1063C58 ETMPSHEL+00258 |  F1031186 NDISERV +02546
  F10058F0 NLCNLMAN+008F0 |  F10053A0 NLCNLMAN+003A0 |  F10450E0 EMMPGSRV+00860
  F1045000 EMMPGSRV+00780 |  F1025320 EMMPGFIX+022A0 |  F10450E0 EMMPGSRV+00860
  F1045000 EMMPGSRV+00780 |  F1025320 EMMPGFIX+022A0 |  F10450E0 EMMPGSRV+00860
  F1045000 EMMPGSRV+00780 |  F1025320 EMMPGFIX+022A0 |  F10450E0 EMMPGSRV+00860
  F1045000 EMMPGSRV+00780 |  F1025320 EMMPGFIX+022A0 |  F10450E0 EMMPGSRV+00860
  F1045000 EMMPGSRV+00780 |  F1025320 EMMPGFIX+022A0 |  F10450E0 EMMPGSRV+00860
  F1045000 EMMPGSRV+00780 |  F1025320 EMMPGFIX+022A0 |  F10450E0 EMMPGSRV+00860
  F1045000 EMMPGSRV+00780 |  F1025320 EMMPGFIX+022A0 |  F10450E0 EMMPGSRV+00860
  F1045000 EMMPGSRV+00780 |  F1025320 EMMPGFIX+022A0 |  F100521C NLCNLMAN+0021C
CMD:
Key: 1=Help 2=Plk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=AUDI   9=Dump
```

Figure 52: Information on AUDIT tables

5.3.2.20 String search (special window: FIND)

The FIND function can be used to search for strings in the entire area of the diagnosis object. The area to be searched can be defined by specifying a memory interval, load units or memory class.
DAMP supports searching with one or two search patterns. If the search involves two patterns, the displacement (offset) from the start of the first pattern to the start of the second patterns must be specified.

The user can specify a special memory alignment with respect to the first search pattern. For both search patterns, DAMP supports the use of various string formats (hexadecimal, character, string and text) and variable wildcard symbols. The search can be restricted by specifying the maximum number of hits.

In order to execute the FIND function, the diagnosis object must be open. The call to search for strings is issued with the statement:

```
START-PATTERN-SEARCH WINDOW=<w>
```

Following the call, the FIND selection window initially appears. This selection window is used to define the search area and the search patterns. You can search within any address space supported by DAMP by making a corresponding entry in the “ASEL” and “ASID” fields (see also page 78):

- in the virtual address space (ASEL = TSN | TID)
- in the data space (ASEL = ALT | SPI)
- in the real address space (ASEL = RM)
- in the absolute address space (ASEL = ABS)
- in the Processor Saved Status (ASEL = PSS)
- in the Hardware System Area (ASEL = HSA)
- in a dumpfile section (ASEL = SCT).

The FIND window is based on the following input principles:

- All entered data is retained on executing the FIND function and serves as the default for the next function to be executed. You can thus always expand on the earlier specification.
- All input fields are interpreted on executing the function exactly a they appear on the screen.
- Before executing the function, only one input field needs to be modified. The only exception is a new FIND window, where at least the search area and the first search pattern must be defined.
- Only blanks may be used to reset inputs to “not specified”. NULL characters (X'00') may not be used for this purpose (except in the case of the two search patterns).

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


FIND - Command                                    TID=00010001          W9,D  ,L19
   Interval     :    Start     =           End     =
or Load Unit    :    Scope     =  ALL       CLASS4    PRIV      NONPRIV   USER
                     Subsystem =           Version =
                     Module    =
or Memory Class :     ALL    C11    C12    C13-PP    C14-PP    C15-PP    C16-FP
                                           C13-FP    C14-FP    C15-FP    C16-MP
                                                     C14-NP    C15-MP

                     Wildcard Symbol = *           Alignment  (B/H/W/D/P)  = H
                     Number of Hits  = 18          Count only (Y/N)        = N
Output Area: *SYSOUT

1.Search Strg C:
Offset :
2.Search Strg C:

Cancel possible with K2 + /INFORM-PROGRAM MSG='CANCEL' (/INTR CANCEL).

CMD:
Key: 1=Help 2=Plk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=Dump   9=FIND
```

Figure 53: Selection mask for searching in the virtual address space

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


FIND - Command                                 ALT=00000000-00010017 W9,D  ,L19
   Interval     :    Start     =           End     =









                     Wildcard Symbol = *           Alignment  (B/H/W/D/P)  = H
                     Number of Hits  = 18          Count only (Y/N)        = N
Output Area: *SYSOUT

1.Search Strg C:
Offset :
2.Search Strg C:

Cancel possible with K2 + /INFORM-PROGRAM MSG='CANCEL' (/INTR CANCEL).

CMD:
Key: 1=Help 2=Plk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=Dump   9=FIND
```

Figure 54: Selection mask for searching within other address spaces (example using ASEL=ALT)

By default the output is directed to the screen, to the same diagnostic window as the input. The user can also specify a file, *SYSLST or *EDT, as the output medium (Output Area). In addition, the user can also only have the number of hits found output and suppress the hit list (“Count only=Y”).

Specifying the search area

When selecting the search area in the virtual address space, you can specify a memory interval (**Interval**), one or more load units (**Load Unit**), or one or more memory classes (**Memory Class**). All three specifications for the search area are mutually exclusive (disjunct), i.e. only one specification can be valid at a given time, and no combinations are supported.
Only an **Interval** can be specified in the other address spaces.

The selection of the currently valid search area occurs in DAMP as follows:
– If precisely one search area is specified - either explicitly or as a default setting from the previous FIND call - that search area is selected.
– If two search areas have been specified, of which only one was specified explicitly, the explicitly specified search area applies.
– If multiple search areas are explicitly specified, the call to the function is rejected with the output of a message.

A search area is considered explicitly specified whenever any changes are not exclusively restricted to overwriting fields with blanks.
After a search area has been selected, all values of the other search areas are implicitly reset by DAMP and do not need to be explicitly reset.

For the virtual address space (ASEL=TSN | TID), the function supports searching in a selected address space (ASID=<tsn>|<tid>) as well as all address spaces (ASID=*ALL) contained in the object. With ASID=<tsn>|<tid>, you can enter search areas from the user and system memory; with ASID=*ALL, only the areas that are fully located in the user address space are allowed.

– **Interval** search area

Start field	Specifies the start address of the search area
End field	Specifies the end address of the search area

The search area is defined fully only when both the Start and End addresses have been specified. There are no default settings.

– **Load Unit** search area
This search area specifies the modules of BLS or DSSM load units.

The delineation of the search area occurs hierarchically in the following levels:

Scope → Subsystem → Version → Module.

If “Load Unit” is set as the search area, it is not necessary to specify all the available fields. An implicit assumption is made for each of the omitted fields.

Scope selection bar

Specifies one or more BLS or DSSM load areas

ALL	All modules (from the CP, all subsystems and the user program)
CLASS4	All modules from the system address space, except CP
PRIV	All modules from CP and the privileged subsystems
NONPRIV	All modules from the nonprivileged subsystems and the user program
USER	All modules from the user program

If “Load Unit” is set as the search area and “Scope” is not specified, “Scope=ALL” is implicitly assumed.
For “NONPRIV” and “USER”, the result of the search depends on whether the set task is connected to the subsystems or whether the task has loaded a program.

Subsystem field

Specifies one or more subsystems
Subsystem names can be entered with a length of up to 8 characters. The use of wildcards to specify multiple subsystems is supported: the “*” symbol matches any number of characters in the name, and “/” matches exactly one character.
If “Load Unit” is set as the search area and “Subsystem” is not specified, “Subsystem=*” (i.e. all subsystems from the specified “Scope”) is assumed.

Version field

Specifies one or more subsystem versions
Versions can be specified with a length of up to 8 characters. The use of wildcards to specify multiple versions is supported (as explained under the “Subsystem” field).
If “Load Unit” is set as the search area and “Version” is not specified, “Version=*” (i.e. all versions of the specified subsystems) is assumed.

Module field

Specifies one or more modules
Module names can be entered with a length of up to 32 characters. The use of wildcards to specify multiple modules is supported (as in the “Subsystem” field).
If “Load Unit” is set as the search area and “Module” is not specified, “Module=*” (i.e. all modules of the specified subsystems) is assumed.

– **Memory Class** search area

This search area specifies one or more memory classes as the search area(s). The memory classes are selected by marking, and all possible combinations are allowed. The selection of memory classes from the user address space always applies only to the currently set task.

The following memory classes and subclasses are supported:

Field		Selected area	
ALL		All memory classes	
CL1	Class-1-Memory	Resident system modules	System address space
CL2	Class-2-Memory	Pageable system modules	
CL3PP	Class-3-Partial-Pages	Resident partial pages	
CL3FP	Class-3-Full-Pages	Resident full pages	
CL4PP	Class-4-Partial-Pages	Pageable partial page	
CL4FP	Class-4-Full-Pages	Pageable full pages	
CL4NP	Class-4-Nonpriv-Pages	Nonprivileged class 4 pages	
CL5PP	Class-5-Partial-Pages	Privileged partial pages	User-address space
CL5FP	Class-5-Full-Pages	Privileged full pages	
CL5MP	Class-5-Memory-Pool	Class 5 memory pool	
CL6FP	Class-6-Full-Pages	Nonprivileged full pages	
CL6MP	Class-6-Memory-Pool	Class 6 memory pool	

Table 7: Memory classes (selected by marking)

Specifying the search patterns

You can specify one or two search patterns. If you are specifying two search patterns, you will need to specify the offset between the start of the first search pattern and the start of the second pattern. Search patterns are specified via the **1.Search Strg**, **Offset** and **2.Search Strg** fields**.**

The **Alignment** field can be used to specify a memory alignment with respect to the first search pattern. In addition, you can also change the **Wildcard Symbol** field.

– **1.Search Strg** field
Specifies the first search pattern. This search pattern must always be specified. You can enter up to 64 characters, which are preceded by on byte to indicate the format type.

The following formats are supported:

X	Hexadecimal format	The allowed characters are 0..9 and A..F
C	Character format	Conversion of lowercase to uppercase in the search pattern, followed by a match
S	String format	No conversion
T	Text format	Conversion of uppercase to lowercase, both for the search pattern and for the search area contents, followed by a match

The default setting is the “C” format.

The use of wildcards in the search pattern is supported. The wildcard symbol corresponds to the character defined in the “Wildcard Symbol” field. It can be used at any position in the search pattern and matches exactly one character at that position.

– **Offset** field
Specifies the offset between the start of the “1.Search Strg” and the start of the “2.Search Strg”. The “Offset” is specified as a hexadecimal value. In order to set the offset to “not specified”, the input field must be overwritten with blanks.

– **2.Search Strg** field
Specifies the second search pattern. The “2.Search Strg” field is defined like the “1.Search Strg” field and is only taken into account in the search if an “Offset” has been specified.

– **Wildcard Symbol** field
Specifies a character to be used as a wildcard symbol in the first and second search patterns (i.e. the “1.Search Strg” and “2.Search Strg” fields). The wildcard symbol is set to “*” by default, but may be modified by the user. All characters except digits, letters and blanks are allowed.

– **Alignment** field
Specifies a memory alignment, with respect to the first pattern (“1.Search Strg”).

The following alignment types are supported:

B Byte boundary
H Half-word boundary
W Word boundary
D Double-word boundary
P Page boundary
The default setting is the “H” format.

A page boundary (“P”) generally means that the search is aligned on a 4 KB boundary. The only exception is when searching in objects that were opened as PAM files, in which case the alignment is on a 2 KB boundary.

Specifying the output for the FIND function

The output medium is defined using the “Output Area” field. By default the output is directed to the screen, to the same diagnostic window as the input. The maximum number of hits for the search is set in the “Number of Hits” field, and “Count Only” defines whether the hit list should be output or just the number of hits. The format of the output can be selected using the “Mode field” in the header line.

– **Output Area** field
Specifies the output medium.

The following output media are supported:

*SYSOUT Screen, same dialog window as for the input
<filename> File
*SYSLST System file *SYSLST
*EDT EDT window which was used last or EDT window 0
The default setting is *SYSOUT.

– **Number of Hits** field
Specifies the maximum number of hits after which the search is to be interrupted and the found hits displayed. The ’Number of Hits‘ is entered as a decimal value. The maximum number of hits displayed is, however, also restricted by the size of the output window.

– **Count only** field
Specifies whether the hit list is to be output or only the number of hits.

The following specifications are supported:

N Hit list
Y Only number of hits
The default setting is Count only=N, i.e. the hit list is output.

– **Mode** field in the header line

  The following output formats are supported:

  D(MP) Normal dump format
  C(HR) Character format
  H(EX) Hexadecimal format
  The default setting is the D(MP) format:

The output format can be entered in the selection and output windows.

Output window of the FIND function

After starting the FIND function with Output Area = *SYSOUT (default setting), the hits are shown in the same diagnosis window. If **one** search pattern was specified, one line is output for each hit; if **two** search patterns were specified, each hit is displayed in two lines, where the second line is always preceded by the “Offset” in the output.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


FIND - Command                                    TID=000A01EB          W8,D  ,L19
724FD008 (ETCB-002+00000) E3C3C240 00010002 C8C5D9E2 00008000 = TCB ????HERS????
724FD016 = * + 0000000E   80000400 00000000 00000000 00000000 = ????????????????
724FD7A8 (ETCB-004+00000) E3C3C240 00010004 C3D3D6C7 00008000 = TCB ????CLOG????
724FD7B6 = * + 0000000E   80000400 00000000 00000000 00000000 = ????????????????
72500008 (ETCB-00D+00000) E3C3C240 0001000D E3C1D7F1 00008000 = TCB ????TAP1????
72500016 = * + 0000000E   80000400 00000000 00000000 00000000 = ????????????????
```

Figure 55: FIND output window (when searching with two patterns)

The search is interrupted when the window is completely filled with hits in accordance with the current window length or when the maximum number of hits set in the “Number of Hits” field is reached. It can then be resumed with +/F3 or aborted with - /F1 .

Furthermore, you can force a cancellation of the search with K2 followed by /INFORM-PROGRAM MSG='*CANCEL'. All hits found up to that point are then displayed.

If the search area includes pages that are not contained in the diagnosis object, this is indicated by messages, but the search is not aborted.

The address of the found search pattern can be marked in each line of the hits and the memory area can be displayed in a dump window, see the section “Marking” on page 86.

In the D and HEX output modes, the individual words in the output area can be marked with MAR and assigned as start addresses to the individual dump windows.

5.3.3 Modification by the user (special window: OPTIONS)

You can set user options for your DAMP application to suit your requirements, regardless of the default settings on delivery or the settings defined by system administration. These settings include the user ID for the path names of the files required by DAMP and other options.

Standard names

All DAMP product files along with their release names and their significance are listed in section “Software and hardware prerequisites” on page 327.

The standard name of a product file is taken to mean the path name provided by IMON.

The standard names of the system symbol library and the system PRODAMP library are the fixed path names $TSOS.SYSSMB.DAMP and $TSOS.SYSDMP.DAMP, respectively.

DAMP always works with these standard names. Other names can be set in a DAMP generated individually (see “Possible parameter settings” on page 134).

Setting user options

DAMP users can set user options specifically to suit their application.
User options are set or changed by the `START-OPTION-DIALOG` command after DAMP is loaded. This command opens a window where the settings can be changed by overwriting and marking. OPTS appears in the key line for the diagnostic window used.

The user options can be changed temporarily each time DAMP is called. The default entry in the “output format” field of the diagnostic window title line is **TMP**.
To make the setting permanent, it is best to first copy the load program SYSPRG.DAMP.<ver>, supplied as a standard DAMP component, at system level to a user-specific file with the name “DAMP”.
Then call DAMP with the new program name, issue the `START-OPTION-DIALOG` statement, set your user options, and overwrite the “output format” field with **SAV**. After pressing DUE, the procedure S.PRC.DAMP.<ver>.OPTIONS is generated. This procedure must be started with `/call-proc s.proc.damp.<ver>.options` when DAMP is terminated; it then modifies the specified load program.

```
DAMP <version>  No Object opened in BS2000 V<ver>  <date> <time>


DAMP user options                                                  W9,TMP,L19

Userids:  SYSLNK / SYSDMP   =  *STD        SYSPAR  (REDUCE)  =  *STD
          SYSSMB            =  *STD        SYSLNK  (ANITA)   =  *STD
          SYSMSH / SYSSDF   =  *STD

Window separation:       yes/no            Window separator          =  -/X'60'
Column separator (screen)   =  |/X'4F'     Column separator (list)   =  |/X'4F'
Trash character             =  ?/X'07'

Message:  Language          =  ENGLISH     Blinking:               yes/no

Lines per list page         =   65

K1 check-back:           yes/no            Save P-Keys:            yes/no

PRODAMP:  Source      =  *STD
          Object      =  *STD

CMD:
Key: 1=Help 2=Plk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=OPTS   9=OPTS
```

Figure 56: OPTS window

Possible parameter settings

The following user options are available in DAMP; the default values set on delivery are shown in bold print.

The value *STD for one of the user IDs specified below means that the relevant standard name (see page 133) is used as a path name for the file it describes.

If a user ID is entered as the value for the field, DAMP searches for the corresponding file only under the entered user ID.

User IDs: SYSLNK/SYSDMP = ***STD** | <userid>

Joint user ID for the DAMP module library and the system PRODAMP library. The system PRODAMP library contains the PRODAMP routines for automatic pre-diagnosis among other things.

SYSSMB = ***STD** | <userid>

User ID for the standard symbol library that is accessed by default when loading the symbols.

SYSMSH / SYSSDF = ***STD** | <userid>

Joint user ID of the auxiliary files, the message files, and the user SDF syntax files containing the DAMP statements.

SYSLNK (ANITA) = ***STD** | <userid>

User ID under which the dynamically loadable library of ANITA is cataloged. Specifying a user ID causes a search to be performed for the SYSLNK.ANITA library under the specified user ID. If an older version of an ANITA library is loaded, the old version is unloaded, and the new version is loaded instead. (The access method ANITA is used by DAMP when accessing dump files and the current system.)

Window separation: **yes** | no

The diagnostic windows are separated from each other with or without dashed lines

Window separator = **- / X'60'**

Window separator for output at the terminal.

Column separator (screen) = **| / X'4F'**

Column separator for output at the terminal because X'4F' is not represented as “|” on all terminals.

Column separator (list) = **| / X'4F'**

Column separator for list output because X'4F' is not output as “|” on all printers.

Trash character = **. / X'07'**

Replacement character for non-printable characters on the terminal. X'00', X'07' and printable characters are possible.

Message: Language = **ENGLISH** | DEUTSCH

Language in which the texts in the help window, the online help texts and the DAMP messages are output. The language can also be set via the help window (W1), see page 60.

Blinking: yes | **no**

Sets the output of DAMP messages in lines 2 and 3 to blinking or not blinking.

Lines per list page = **65**

Defines how many lines are to be printed on a page in a list output.

K1 check-back: **yes** | no

Defines whether DAMP should be terminated immediately after the K1 key is pressed or whether the system requests users to confirm their entry

Save P-Keys: **yes** | no

Defines whether the P keys are saved by DAMP before they are overwritten, and restored by DAMP after interruptions or termination This option is only evaluated on some terminals (firmware program version FW_976x=X'20').

PRODAMP: Source = ***STD** | <filename>

Library in which the user source programs are stored.

Object = ***STD** | <filename>

Library in which the user objects are stored.
In both cases, *STD stands for the SYS.USRDMP.DAMP.<ver> library of the execution user ID.

A temporary setting for the user PRODAMP libraries does not take effect immediately. It is merely a default value for the *STD value in the SOURCE-LIBRARY and OBJECT-LIBRARY operands of the DAMP statement ASSIGN-PRODAMP-LIBRARIES. If the settings are to take effect immediately, this statement should be issued after setting the user options.

5.3.4 Additional functions

5.3.4.1 Calling EDT as a subroutine

The EDIT-FILE statement can be used to call EDT as a subroutine, thus making the EDT functions available (see the “EDT” manual [2]). This enables additional document files such as CONSLOG, SERSLOG, HERSFILE, etc. to be used for on-screen diagnosis in parallel with the processing of the dump file in the diagnostic windows .

DAMP itself also uses EDT for certain functions:

– the descriptor list from the automatic preanalysis is stored in EDT area 8

– system files and dump sections can be processed with EDT in special FILE windows

– the diagnostic language PRODAMP uses EDT for editing procedures and for output from procedures which are running

All EDT functions except the EDT statements @LOAD and @EXEC are available in EDT. These statements are always rejected.

At the EDT program level, messages from EDT and DAMP are displayed in the last one or two data lines on the EDT screen. If this causes some of the screen contents to be lost, the original status can be restored by pressing the K3 key.

Normally, the K1 key is used to return to the DAMP program level, but the same effect can be achieved by entering HALT or END in F mode and @HALT or @RET in L mode.

The HALT statement can also be entered with the following operands:

HALT @	Output of the current EDT statement symbol in a DAMP message line.
HALT msg	Output of the string “msg” in the DAMP message line.
HALT #msg	Output of the string “msg” in the DAMP command line.

This also applies to @HALT and @RET (but not END).

5.3.4.2 Logging and replaying a diagnostic session

If desired, all screen inputs and outputs can be logged, i.e., saved to a file.

- The logging file can be edited for printing with the `PRINT-LOGGING-FILE` statement. The actual printing is also initiated with this command (see description of the statement on page 202).
- The diagnosis steps can, for example, be replayed and checked by a different person.
- The diagnosis steps performed up to now can be replayed by the same person if the diagnosis run was interrupted either on purpose or unexpectedly.

Logging a diagnostic session

Logging of a diagnostic session is activated at program level by using the `LOG-SESSION` statement and at system level by using the BS2000 command:
`INFORM-PROGRAM MSG='*LOG-SESSION'`

The name of the logging file is specified at program level with the statement `LOG-SESSION LOGGING-FILE=filename` or at system level with the BS2000 command:
`ADD-FILE-LINK LINK-NAME=DAMPLOG,FILE-NAME=filename`

After the logging file has been closed, the link name DAMPLOG is released.
If this is not done, the file name is generated automatically in the form
`S.LOG.DAMP.<ver>.<date>.<time>`.

Logging is terminated at program level with the `STOP-LOGGING` statement and at system level with the BS2000 command: `INFORM-PROGRAM MSG='*STOP-LOGGING'`

If the problem is being passed on to another diagnostic technician for further analysis, the logging file should be included in file form.

Printing a diagnostic session

The diagnostic log is printed with the `PRINT-LOGGING-FILE` statement (see description of the statement on page 202).

Replaying a diagnostic session

Any logging file created with DAMP can be replayed with DAMP by the person who created it or by any other DAMP user.

Replay of a diagnostic session is started at program level using the statement `REPEAT-SESSION <loggingfilename>` and at system level using the command `INFORM-PROGRAM MSG='*REPEAT-SESSION(<loggingfilename>)'`.

All inputs and outputs of the logged DAMP dialog are displayed on the screen and each must be acknowledged by means of DUE or K3.

Pressing the K2 key causes a switch to system mode. If you are already in system mode, you can return to the replay function with the `RESUME-PROGRAM` command.

K1 can be used to terminate the replay function prematurely.

If the entire logging file is replayed, the system returns to the level at which the replay function was started.
If the replay function was started by means of `REPEAT-SESSION <loggingfilename>`, the DAMP screen mask will be displayed at the end of the replay.

If a replay is started from system mode with the command `INFORM-PROGRAM MSG='*REPEAT-SESSION(<loggingfilename>)'` you will be returned to system mode at the end of the replay. In this case, diagnosis with DAMP can be resumed with the `RESUME-PROGRAM` command.

5.3.4.3 Processing files in PAM format

With certain restrictions, it is also possible to analyze files which are not stored in the BS2000 dump format. This function is provided primarily intended for an “emergency analysis” of damaged dump files.

The statement `OPEN-DIAGNOSIS-OBJECT filename(KIND-OF-OBJECT=*PAM)` can be used to open any disk file in PAM format. The file itself can, of course, also be a SAM or ISAM file.

During processing, all DAMP functions which do not require the file to be diagnosed to have a normal BS2000 structure can be used, i.e. it is possible

– to display PAM pages of the file in the usual formats (D, H, C, ...) in various dump windows

– to (manually) assign any symbol file for symbolic editing

– to search for strings (`START-PATTERN-SEARCH`) using wildcards, with restriction of the search area also being possible

– to output edited PAM page ranges to SYSLST

– to use procedures written in the diagnostic language PRODAMP.

With respect to addressing, PAM files differ from BS2000 files in the following respects:

– PAM page numbers

  PAM page numbers are used instead of the module-relative addresses normally used for BS2000 dumps. The PAM page is entered in the form P-XXXXXX (hexadecimal page number) as of column 1 of the title line of a dump window. The first page of a file is the page P-000001.

– Absolute addresses

  For absolute addressing, the entire file is regarded as an unstructured “stream” of bytes. The absolute address numbers these bytes (starting with 0) throughout the file. The absolute address of the first byte on page P is thus A = (P - 1) * 2048. Absolute addresses may be entered in column 40 of the title line of a dump window.

When address fields are marked, addressing via PAM pages is used, i.e. the rightmost three bytes of the marked word are interpreted as a page number and this page is assigned to the appropriate window. This corresponds, for example, to the method used to represent chaining in dump files via PAM pages.

When memory segments are output to SYSLST, only entire PAM pages are output. For this reason, page numbers (without “P-”) must be entered in LIST windows.

There are two ways of searching for strings:
– within a single page whose page number is specified
– as absolute addresses within a segment whose limits are specified.

In the case of large files, the length of a data section may exceed the 4-GB boundary. Since the absolute addresses used in DAMP cannot be greater than this, an internal segment number is used to distinguish between the
4-GB segments. When a PAM page number is specified, the correct segment is selected automatically. The absolute addresses are then relative to the start of the segment. With `START-PATTERN-SEARCH` , it is possible to enter a segment number for the search explicitly.

The stack window (W3) and most of the function-specific windows of DAMP assume that the object to be diagnosed has a BS2000 structure. Consequently, these windows cannot be used to process files in PAM format.

When processing PAM files, the status window (W2) contains information on the currently open file itself, e.g. the file size and the last-page pointer.

5.3.4.4 Editing SLEDs without a BS2000 structure

Using DAMP, SLEDs produced on an operating system other than BS2000 (e.g. IPL, BOOT, STARTUP or SLED) can also be processed.

Any dump file can be opened without virtual addressing with the statement
`OPEN-DIAGNOSIS-OBJECT <filename> (KIND-OF-OBJECT=*SELF-LOADER).`

DAMP does not offer an automatic edit function for the editing of files opened as self-loaders. All addresses are interpreted as real addresses. The areas in main memory can only be accessed via real addresses.

The following functions are possible:

– output of memory pages in the usual formats (D, H, C, ...) to various dump windows
– (manual) assignment of any symbol file for symbolic editing of the output
– selective search for strings (FIND function) using wildcards, with the option of restricting the search area
– output of page areas to SYSLST in the normal DAMP layout
– use of PRODAMP procedures that permit the easy analysis of SLEDs without a BS2000 structure

5.3.4.5 Using private symbol elements

DAMP is supplied with the standard library SYSSMB.DAMP.<ver>, which is merged into the library $TSOS.SYSSMB.DAMP. This library contains the most frequently used DSECTs. An overview of the DSECTs can be found in section “List of DSECTs from the standard symbol files” on page 330.

It is also possible to generate, extend or modify symbol elements and to then assign them for the diagnosis. Typical examples are:

– DSECT tables for DCM

– DSECTs for the data structures used in a TU program (for evaluating user dumps generated by this program).

On opening the diagnosis object, DAMP automatically assigns the matching BS2000 system version symbol element.
This automatic function can be disabled by explicitly specifying a symbol element in the OPEN-DIAGNOSIS-OBJECT statement. This symbol element will then be used to process the object to be opened. You should, however, note that the standard BS2000 symbols must be included in this object.

Additional symbol elements can be assigned using the ADD-SYMBOLS statement. When a DSECT is subsequently specified, all assigned symbol elements are searched for the matching information, starting with the symbol element most recently entered.

All assignments are reset on switching the dump file.

Generating private symbol elements

You can generate your own symbol element in the following manner:

– Assemble the additional or modified DSECTs with TEST-SUPPORT=*AID or with *COMOPT ISD (a dummy CSECT should be added to the source code after the last DSECT since the Assembler will otherwise calculate the length of the last DSECT incorrectly).

  In the case of SPL models, the option *COMOPT SYMTEST=ALL must be specified for the compiler.

  If a module containing symbol information already exists, a new compilation run is not necessary.

  C structures must be compiled using TEST-SUPPORT=YES. A pointer must be defined for each symbol (=type) to be generated, since the C compiler only stores the name of the variable. Structures and arrays are supported in this manner. The pointers should be defined in the same sequence as the structures to which they refer. Only in this manner can the reference between the structures and the pointers be analyzed, this also saves memory space. When performing a search for a symbol, DAMP does not differentiate

between uppercase and lowercase which means that the sole distinction between the names of main structures (DSECTs) must not be in the form of uppercase and lowercase letters. For this reason, the symbol generator checks the generated main structures for uniqueness, keeps the first relevant structure it finds, and eliminates the next one.

– After the statement `/START-DAMP-SYMBOL-GENERATOR` has been issued, the system queries whether symbols are to be generated (enter “g”) or whether symbol information is to be output (enter “i”). If you enter “g”, the system then asks for the module and library containing the system information. The symbol information is then stored in a PLAM library as a type X element. The system queries the name of the library and the element.

  The library for standard BS2000 symbols has the fixed name `$TSOS.SYSSMB.DAMP`. The element name is the same as that of the product to which the symbols refer, and the element version is likewise derived from the product version (e.g. BS2000/200 for BS2000 V20.0A = BS2000 OSD/BC V11.0)

  If there is already a symbol library with the same name under the active user ID, the newly compiled DSECTs can be included in this library. If a symbol element with the same name already exists in the specified library, you can choose either to replace it or to supplement it with the new information.

– If required, copy the symbol library or symbol element under the desired user ID or into the desired library and set the user option “SYSSMB” (see page 134).

The following two examples illustrate the use of `/START-DAMP-SYMBOL-GENERATOR`.

Example 1

```
/start-damp-symbol-generator
You wish to : - Generate symbols ?                              -->    g

              - Get information about symbols ?                 -->    i

*g
  Creation of a DAMP-Symbolfile.
  Please give name of : - library with object module or
                        - old symbol file for conversion
*my.object.lib
  Please enter name and type of object module
  (e. g. 'MODNAME/R' [type R is default])
  In C it is the R-element with "@" as termination.
  In SPL it is the 8 B long R-element with "@" as termination.
*dmpbs2a/r
  Element  DMPBS2A/@/R
  from library  MY.OBJECT.LIB  successfully opened.
  Symbolic information will be taken from LSD-cards.
  The symbol information is from BS2000 V190, PVLU E1.
  Proposal : The element BS2000/190.E1
             will be generated in the library
             SYSSMB.DAMP

  Please enter one of the following answers :

  Y[ES]      -> You accept the proposal.
  L[IBR]     -> You will further be asked for the name
                of the output library where the element
                BS2000/190.E1
                will be generated.
  U[SER]     -> By user, you will further be asked
                for the name of the output library and
                for the name and version of the element.
  P[ROD]/N[O]-> You will further be asked for the name,
                the version and the PVLU of the product.
                Lib : SYSSMB.<prod>.<vers>.<PVLU>
                El  : <prod>/<vers>.<PVLU>
  I[NPUT]    -> Output library/element/version =
                Input  library/element/version.
                Output element type = X.
*u

  Please enter valid names  !!!

  1. -> Enter library name    :
*my.symbol.lib
  2. -> Enter element name    :
*my_element
  3. -> Enter element version :
*190
  Output Symbol Library : MY.SYMBOL.LIB
  Output Symbol Element : MY_ELEMENT/190

  Element  MY_ELEMENT/190 from
  library  MY.SYMBOL.LIB  successfully opened.
  Starting to write symbol element.
  There are to be generated 75 structures.
  Symbol element written.
  There have been written 75 structures into the file.
  Program terminated normally.
```

Example 2

/START-DAMP-SYMBOL-GENERATOR can also be used to have the DSECTs, structures and symbols stored in symbol elements listed on the screen or saved in a file.

```
/start-damp-symbol-generator
  You wish to : - Generate symbols ?                              -->    g

                - Get information about symbols ?                 -->    i
*i
Please enter the library name  :
*syssmb.damp.<ver>
You wish to : - list the library elements and the contained symbols
                on screen ?                                       -->    s
              - write the names of the DSECTs into a file ?       -->    n
              - write a DSECT, converted to a Pascal-Record,
                into a file ?                                     -->    r
              - search for a symbol with wildcards ?              -->    w
              - write the alphabetical list of symbols into EDT-->       a
              - show the global info of a symbol element          -->    g
              - go to EDT ?                                       -->    @
              - assign a new library ?                            -->    l
              - terminate the program ?                           -->    e
*s
The library SYSSMB.DAMP.<ver> contains the following elements :
BS2000/180        BS2000/190        BS2000/200
BS2000-USER/180   BS2000-USER/190   BS2000-USER/200
NSDIO/180         NSDIO/190         NSDIO/200
STATUS/000        STATUS/001        STATUS/002        STATUS/003
STATUS/004        STATUS/005        STATUS/006        STATUS/007
STATUS/008        STATUS/009        STATUS/010        STATUS/011
XA2000/180        XA2000-USER/180   XA2000/200        XA2000-USER/200
XA2000/190        XA2000-USER/190
You wish to : - list the symbols     --> name/version
              - stop this function   --> *e, *end
*BS2000/200
  ASAVDSSM                          ASIMDBHD
  ASIPUCON                          BS_CTX_VECTOR_REC_MDL
  CTX_VECTOR_REC_MDL                DBL_OPTIONS_COM_MDL
  DBL_OPTIONS_P_C_MDL               DBL_OPTIONS_S_P_MDL
  DSTE                              DWQE
  DWQH                              EBWL
  ECSA                              ECSE
  ECSX                              ECTLP
...

You wish to : - list the library elements and the contained symbols
                on screen ?                                       -->    s
              - write the names of the DSECTs into a file ?       -->    n
              - write a DSECT, converted to a Pascal-Record,
                into a file ?                                     -->    r
              - search for a symbol with wildcards ?              -->    w
              - write the alphabetical list of symbols into EDT-->       a
              - show the global info of a symbol element          -->    g
              - go to EDT ?                                       -->    @
              - assign a new library ?                            -->    l
              - terminate the program ?                           -->    e
*e
Program terminated normally.
```

5.3.4.6 Writing private Assembler user routines

If the situation demands it, you can write your own user routines for list editing or for the special evaluation of dumps. You can then call these routines from DAMP by means of the LOAD-MODULE and START-MODULE statements. However, this makes you dependent on the structure of the dump file and on the BS2000 version being used. These dependencies are dissolved if you use the diagnostic language PRODAMP instead. Within PRODAMP, you can call Assembler routines using the PRODAMP function ENTER-MODULE. This has the advantage of allowing you to transfer diagnostic data as parameters.

When writing private user routines the interface must comply with the following conditions:

Register 1	contains the address of the parameter string (up to 80 characters) which can be specified in the START-MODULE statement.
Register 13	contains the address of an 18-word save area which is made available by DAMP and which can be used in accordance with the VMOS (Virtual Memory Operating System) conventions.
Register 14	contains the return address.
Register 15	contains the entry address.

All registers must be reset to their original values before control is returned to DAMP.

The module called cannot have any specific requirements regarding the status of the diagnosis object currently being processed under DAMP. Neither does DAMP provide any interfaces which can be used by external routines. Moreover, the external procedures are called in 31-bit mode, which means that they must contain at least one 31-bit adapter. The user module may be stored in any module library. Before the routine is called via START-MODULE, the module must be loaded dynamically by means of a LOAD-MODULE statement.

If the user module is stored in the dynamically loadable library of DAMP, the START-MODULE statement can be omitted.

In the case of teleprocessing problems, you can initiate the DCM user routine, which provides you with edited DCM tables.

5.4 Generating and printing lists (special window: LIST)

In spite of the convenience provided by on-screen diagnosis, it is often useful to print the dump on paper. If, for example, you have used PRODAMP for special evaluations and to group together certain structures which are normally scattered throughout memory, you can print the results on a printer of the local computer or on an external computer. You can, of course, also print the entire dump, but the resulting stack of paper can be up to one and a half meters high and has little value, except for demonstrating how big a dump can be, compared with the ease of diagnosis offered by DAMP.
Editing lists for printing can be controlled in interactive mode or by means of batch statements.

5.4.1 Controlling list output in interactive mode

When the START-LIST-GENERATION statement is entered, the list mask is displayed in the last free diagnostic window. If a dump file has already been opened, the name of this file is displayed in the field **Dumpfile**. If there is no dump file open, or if some other dump file is to be edited for printing, you must first select a file (see page 148). The extent and the contents of the list to be printed are then defined by marking or filling out the various fields in the mask.
If an area dump is being processed, the specification of selection criteria has no effect since only the standard tables belonging to an area dump and the requested segments are edited.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


LIST - Command                                  SYS=0                W8,LST,L19

Dumpfile = :SLED:$DUMPFILE.SLED.CS507K
Listfile = *SYSLST                                              REMOTE: YES/NO

FUNCTION : OPN/LST/LSTALL/RESET           SELECT : INF/SYS/MIN/ALL OR
           ------------------------------------------------------------------
SELECT FROM |TRACES: ALL   STT   TM    NO                                    |
            |MAPS  : ALL   CS    CRI   NO                                    |
            |TABLES: ALL   XVT   TCB   PCB     SPL        TFT        AUD     NO|
            |MEMORY: ALL   CL1   CL2   CL3     CL4        CL5        CL6     NO|
            |                          PP FP   PP FP NP   PP FP MP   FP MP     |
            |MODULE:                                                          |
           ------------------------------------------------------------------
            |PAGES         FROM:       TO:                                   |
           ------------------------------------------------------------------
WINDOW:
DIAG: YES/NO   DESCR: YES/NO              PROC:

CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=LIST   9=Dump
```

Figure 57: List mask

The list mask remains displayed on the screen during and after processing of the print job. The print job does not close the last dump file which was processed, and diagnosis of this dump can be continued. Switching to another diagnostic window interrupts LIST processing. You can return to the list mask by pressing the appropriate P key or by entering a new `START-LIST-GENERATION` statement.

If no further editing is desired, the window occupied by the list mask can be released for other outputs by means of the statement `SHOW-EDITED-INFORMATION *STORAGE-EDIT`.

The list mask appears on the screen only in interactive mode. The current settings are displayed with increased intensity, i.e. marking a field and pressing the DUE key causes the marked field to be highlighted and any alternative fields (as in the case of SYS/MIN/ALL) are displayed with normal intensity.

5.4.1.1 Selecting a file

The desired file is selected by entering a fully or partially qualified file name in the field “Dumpfile” of the list mask. Wildcards may be used in the file name. In addition, the string “$TSN” within the file name is replaced by the TSN of the calling task; this is particularly useful for locating currently generated user dumps.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


LIST - Command                                  SYS=0                W8,LST,L19

Dumpfile = :SLED:$DUMPFILE.SLED.CS507K
Listfile = *SYSLST                                                REMOTE: YES/NO
```

Figure 58: Fields for file selection

If wildcards are used in the file name, sending off the modified window with DUE will cause an internal list of matching file names to be created and the first of these names to be displayed in the list mask. You can page forwards and backwards within the list of file names with F3 / + or F1 / - until you find the desired dump file(s).

This file selection process has no effect on any currently open dump file. The currently open dump file will be closed, and the selected file(s) opened, only when one of the functions OPN, LST or LSTALL is marked.

5.4.1.2 Selecting the output location of the list

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


LIST - Command                                  SYS=0                W8,LST,L19

Dumpfile = :SLED:$DUMPFILE.SLED.CS507K
Listfile = *SYSLST                                                REMOTE: YES/NO
```

Figure 59: Fields for the output location of the generated list

The REMOTE option and the “Listfile” field are provided for specifying the output location of a generated list. By default, the list is output to SYSLST on the same computer.

The output file can be directed to a remote computer by marking the REMOTE option and using file transfer.

YES If YES is marked, new input fields are offered in lines 7 and 8 in which the options for the file transfer must be specified (see figure 60). There is no default for these options.

The name of the output file to be sent via file transfer is generated automatically by DAMP (SYSLST.DAMP.<ver>.<date>.<time>). The name contains the current time stamp so that no file is corrupted on the target user ID.

If there is no file transfer connection active, the target computer must be entered in the “Partner” field. The file transfer authorization can be specified using the name of an FTAC profile or explicitly by means of a user ID, account number and password.

NO If NO (default value) is marked, the list is output on the same computer. The name of the output file can be entered in the “Listfile” field for this.

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


LIST - Command                                  SYS=0                W8,LST,L19

Dumpfile = :SLED:$DUMPFILE.SLED.CS507K
Partner  =             FTAC =                                  REMOTE: YES/NO
Userid   =          Account =          Password =
```

Figure 60: Options for file transfer (after marking REMOTE: YES)

5.4.1.3 Selecting a function

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


LIST - Command                                  SYS=0                W8,LST,L19

Dumpfile = :SLED:$DUMPFILE.SLED.CS507K
Listfile = *SYSLST                                                REMOTE: YES/NO

FUNCTION : OPN/LST/LSTALL/RESET           SELECT : INF/SYS/MIN/ALL OR
```

Figure 61: Function selection options

Marking one of the alternatives listed after FUNCTION and sending it off with DUE determines what is to be done with the selected dump file:

OPN — The dump file specified after **Dumpfile** is opened. Any currently open dump file is closed before this is done. This makes it possible, before starting the interactive list output, to check that the correct dump file has been selected. The OPN field can generally be used for opening dump files (instead of using the OPEN-DIAGNOSIS-OBJECT statement).
In the case of dump files with multiple objects (complete VM2000 SLED file, SLED from a SLED / Dump from a SLED), in contrast to using the OPEN-DIAGNOSIS-OBJECT statement to open dump files in interactive mode, the object to be analyzed is selected automatically here.

LST — The dump file specified after “Dumpfile” is printed. All high-intensity parameters in the list mask and all marked parameters are thereby enforced.

LSTALL — All dump files (with a BS2000 object) contained in the file list are printed. The parameters highlighted in the list mask and the marked parameters apply to each of these files.

RESET — All parameters in the list mask are reset to their default values. At the same time, any existing list of dump file names is deleted.

The functions OPN, LST and LSTALL close any dump file which is currently open if the output is not to be taken from this file.

5.4.1.4 Selecting a task

You select a task in a SLED or SNAP by overwriting the “SYS” field in the LIST mask. You can enter the permitted combinations of ASEL and ASID (see page 78).
If a task is specified, only the task-specific areas of this task are output (as if a system dump had been generated for this task).

In the LIST mask it is also possible to enter the keyword ***ALL** or ***ERR** when entering a TSN.

*ALL For a SLED or SNAP file, task editing is performed for all active tasks instead of just the areas of the error task selected by DAMP, as is normally the case.

*ERR causes a search to be started for the possible error task by means of the DIAG pre-diagnosis routine. This entry is equivalent to marking YES in the DIAG field (see page 156).

5.4.1.5 Specifying the scope of the list

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


LIST - Command                                    SYS=0                W8,LST,L19

Dumpfile = :SLED:$DUMPFILE.SLED.CS507K
Listfile = *SYSLST                                                  REMOTE: YES/NO

FUNCTION : OPN/LST/LSTALL/RESET           SELECT : INF/SYS/MIN/ALL OR
           ------------------------------------------------------------------------
SELECT FROM |TRACES: ALL   STT   TM    NO                                        |
            |MAPS  : ALL   CS    CRI   NO                                        |
            |TABLES: ALL   XVT   TCB   PCB     SPL        TFT        AUD     NO|
            |MEMORY: ALL   CL1   CL2   CL3     CL4        CL5        CL6     NO|
            |                          PP FP   PP FP NP   PP FP MP   FP MP     |
            |MODULE:                                                           |
           ------------------------------------------------------------------------
            |PAGES         FROM:       TO:                                     |
           ------------------------------------------------------------------------
WINDOW:
DIAG: YES/NO   DESCR: YES/NO              PROC:

CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=LIST   9=Dump
```

Figure 62: Global selection parameters for the scope of the list

The scope of the list is determined by marking the SELECT fields. These permit the selection of four global output formats (**INF**/**SYS**/**MIN**/**ALL**) and of the selected output of certain user-specified areas (**OPT**). In the first case, DAMP decides which areas are output; in the second case, you decide. In order to make the difference between these two possibilities more obvious, only the detailed selection parameters permitted in connection with **SELECT FROM** are shown within the inner frame of the list mask. Filling out or marking any field within this frame causes an automatic switch to SELECT FROM.

The parameters for the global outputs have the following meanings:

INF Information edited in INF mode in the status window is output to SYSLST.

SYS Only the system overview is output. Depending on the settings of the **DIAG** and **DESCR** parameters, this may include output of the error descriptors (see page 156).

MIN A minimum list is output (see page 160). This list generally contains all the data necessary for an initial diagnosis. If necessary, any gaps can be filled by selecting additional explicit lists (SELECT FROM). This is the default value.

ALL A complete list is output (see page 160). For system and user dumps, this means that all pages saved by CDUMP are included in the list.

The complete list of a system dump produces a stack of paper about one and a half meters high.

Otherwise, only those areas selected explicitly are output within the frame. An explanation of the options can be found in section “Selecting individual areas for output” on page 153.

5.4.1.6 Selecting individual areas for output

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


LIST - Command                                  SYS=0                W8,LST,L19

Dumpfile = :SLED:$DUMPFILE.SLED.CS507K
Listfile = *SYSLST                                                REMOTE: YES/NO

FUNCTION : OPN/LST/LSTALL/RESET           SELECT : INF/SYS/MIN/ALL OR
            ------------------------------------------------------------------
SELECT FROM |TRACES: ALL   STT   TM    NO                                      |
            |MAPS  : ALL   CS    CRI   NO                                      |
            |TABLES: ALL   XVT   TCB   PCB     SPL         TFT         AUD     NO|
            |MEMORY: ALL   CL1   CL2   CL3     CL4         CL5         CL6     NO|
            |                          PP FP   PP FP NP    PP FP MP    FP MP     |
            |MODULE:                                                             |
            ------------------------------------------------------------------
            |PAGES         FROM:       TO:                                       |
            ------------------------------------------------------------------
WINDOW:
DIAG: YES/NO   DESCR: YES/NO              PROC:

CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=LIST   9=Dump
```

Figure 63: Individual areas that can be marked for output

The parameters shown within the frame are used for the explicit selection of individual areas for output. Filling out or marking any field within the frame causes an automatic switch to SELECT FROM. Conversely, marking any option outside the SELECT frame deactivates the settings within this frame. However, the settings are not “forgotten” and can be activated again by marking SELECT FROM.

Some of the fields are used to specify the memory area by marking or entries. A task must be selected for these so-called memory options when analyzing SLED or SNAP files and specifying task-local memory.

The following options are available:

TRACES — This specifies which system trace is to be output:

ALL	all traces
STT	the system trace table
TM	the overview of the traces managed by the trace manager
NO	no trace

MAPS — This specifies which information for the localization of addresses and corrections is to be output:

ALL	the CSECT maps and the Rep information
CS	only the CSECT maps
CRI	only the contents of the SAVEREP or REPLOG section
NO	no maps

TABLES — This selects the system tables to be output.
The output has the same format as for the implicit output of these tables as part of a minimum or full list output. The required tables are selected by marking their names. If task-local tables (TCB, PCB, etc.) are marked in a SLED file, a task must also be selected (see section “Selecting a task” on page 151).

MEMORY — This defines the precise limits of memory segments which are to be output in dump format in the list. Whereas the standard list output (= minimum) contains only memory pages referenced from PCBs or save areas via general-purpose registers, this option permits the specification of contiguous memory segments.
This can be done by simply marking the abbreviations of the memory segments to be output (e.g. CL1) transferring them with DUE. The abbreviations are interpreted in exactly the same way as when searching for strings (see page 125).

For all MEMORY parameters, a task must also be selected when analyzing SLED or SNAP files and specifying task-local memory (see section “Selecting a task” on page 151).

The following are additional memory options:

PAGE FROM Defines a page as a lower limit for the memory area to be output; the contents of the ASID input field is taken into account for real/absolute addresses (ASEL=RM/ABS).

PAGE TO Defines a page as an upper limit for the memory area to be output; the contents of the ASID input field is taken into account for real/absolute addresses. Page numbers (up to 8 hexadecimal digits) are within the range of X'0' to X'1FE00000'.

WINDOW Specifies a standard dump window whose contents (a memory area) are to be printed. In order to list the contents of a window, the window must be filled, which means that a dump file must be opened. Consequently, LSTALL cannot be used in combination with this option. In the case of symbolic output, the output extends to the end of the DSECT; for all other output formats, it extends to the next page boundary. It may thus be necessary to increment the output address and repeat the output in order to obtain all the information.
The following window formats can be output to SYSLST:
- memory areas in dump, hexadecimal and character format
- memory area in Assembler format (disassembled)
- output in real and absolute addressing mode
- symbolic output
- hardware information

MODULE Specifies a system module that is to be output in its entirety.

5.4.1.7 Fields for pre-diagnosis and error descriptors

```
DAMP <version> SLED(<ver>) from BS2000(<ver>)       <date> <time>


LIST - Command                                  SYS=0                W8,LST,L19

Dumpfile = :SLED:$DUMPFILE.SLED.CS507K
Listfile = *SYSLST                                                REMOTE: YES/NO

FUNCTION : OPN/LST/LSTALL/RESET           SELECT : INF/SYS/MIN/ALL OR
           ------------------------------------------------------------------
SELECT FROM |TRACES: ALL   STT   TM    NO                                     |
            |MAPS  : ALL   CS    CRI   NO                                     |
            |TABLES: ALL   XVT   TCB   PCB     SPL        TFT        AUD     NO|
            |MEMORY: ALL   CL1   CL2   CL3     CL4        CL5        CL6     NO|
            |                          PP FP   PP FP NP   PP FP MP   FP MP     |
            |MODULE:                                                          |
            ------------------------------------------------------------------
            |PAGES         FROM:       TO:                                    |
            ------------------------------------------------------------------

WINDOW:
DIAG: YES/NO   DESCR: YES/NO              PROC:

CMD:
Key: 1=Help 2=Tsk 3=PCB 4=Dump   5=Dump   6=Dump   7=Dump   8=LIST   9=Dump
```

Figure 64: Marking fields for the list output

Additional parameters for list output are defined in the DIAG and DESCR fields. Either YES or NO can be specified for each of these options.

DIAG The default value for this field is NO.
If YES is marked, automatic pre-diagnosis is started. This marks a number of memory pages which are subsequently output as part of a minimum list. Automatic pre-diagnosis is not yet supported for user dumps.

DESCR The default value for this switch is NO.
If YES is marked, a list of error descriptors is output in the system overview. DIAG=YES is a prerequisite for DESCR=YES.
On the other hand, it can be useful to perform automatic pre-diagnosis without outputting descriptors since pre-diagnosis references a number of memory pages which are automatically output as part of any subsequent minimum list.

Any illegal combination of the DIAG and DESCR fields is automatically changed to a valid combination.

5.4.1.8 Using PRODAMP procedures or editing programs

The name of a PRODAMP procedure or an editing program can be entered in the PROC field, regardless of which area is selected. The procedure or the program is started once editing of the list is complete, and the lists are generated in addition to those selected in the SELECT frame. The procedure or the program must be available in compiled form as an object in the user PRODAMP library. The library may have to be set by means of the `ASSIGN-PRODAMP-LIBRARIES` statement.

5.4.1.9 Using an editing program

The DMP_ANALYSE_DMS_TABLES program

By entering *DMS in the PROC field of the LIST mask, special DMS editing can be initiated, regardless of the error components detected by the automatic preanalysis. This contains the task-specific DMS tables (TFT, TPR-FCB, TU-FCB) of the relevant task.
The editing is performed via the PRODAMP procedure DMP_ANALYSE_DMS_TABLES, which is supplied with DAMP as part of the standard package.

The MEMCNTRL program

If problems occur in terms of memory assignment, address space bottlenecks, etc., you can start the analysis of the address space assignment by using the MEMCNTRL program within the DAMP application. The MEMCNTRL program is supplied in the system PRODAMP library and must be set with the statement
`ASSIGN-PRODAMP-LIBRARIES OBJECT-LIBRARY=PRODAMP-SYSTEM-LIBRARY`
before calling the program.

The NDM program

In the case of problems from the scope of device management, you can start editing procedures for the NDM tables from within the DAMP application and have the analyses output to SYSLST by making an appropriate entry in the PROC field of the LIST mask. For more information on this topic, see also the chapter “NDMDAMP Generating diagnostic documents” on page 339.

As in the case of the MEMCNTRL program, the system PRODAMP library must be set as the OBJECT-LIBRARY before calling the NDM program.

The output to a list is usually not sufficient documentation when forwarding an error message. In general, the dump file should also be supplied on a data medium for all error messages to permit a diagnosis with DAMP for subsequent instances as well.

5.4.2 Controlling list output in batch or procedure mode

There are three stages involved in controlling the scope of a list:

1. start list generation with the START-LIST-GENERATION statement
2. select memory areas with the ADD-LIST-OBJECTS and REMOVE-LIST-OBJECTS statements
3. start list output using the PRINT-LIST statement

Any number of ADD-LIST-OBJECTS and REMOVE-LIST-OBJECTS statements can be inserted in any sequence between the START-LIST-GENERATION and PRINT-LIST statements. The selected areas are added together and take effect only when the final PRINT-LIST statement is issued.

The START-LIST-GENERATION statement contains all the specifications regarding the input medium, in other words the names of the dump files to be evaluated. The PRINT-LIST statement defines the output medium (SYSLST or file).
Please refer to the relevant statement descriptions for details.

Every list generated by DAMP includes at the end of the report a list of the objects selected in the form of ADD-LIST-OBJECTS statements. Thus, for instance, the required options can be set interactively by marking them and then the corresponding statements can be taken from the generated list.

Examples of statement sequences

The following examples are intended to illustrate the possible structure of a statement sequence in batch or procedure mode in DAMP.
The statement sequences, which start with START-LIST-GENERATION and end with PRINT-LIST, could, of course, be stored in a file, which is then assigned at execution time using the DAMP statement START-STATEMENT-SEQUENCE.

Example 1

```
/BEGIN-PROCEDURE
/ASSIGN-SYSDTA TO-FILE=*SYSCMD
/MODIFY-JOB-SWITCHES ON=5
/START-DAMP
START-LIST-GENERATION FILES-TO-EVALUATE=DUMP.HUGO ————————————————————  (1)
ADD-LIST-OBJECTS TASK-INFORMATION=*PARAMETERS(SELECT=C'RP01', -
                 INFORMATION=*PARAMETERS(CONTROL-BLOCKS=*TCB, -
                 PAGES=*INTERVAL(FROM=X'0',TO=X'FFF'))) ———————————————  (2)
ADD-LIST-OBJECTS GLOBAL-INFORMATION=*PARAMETERS(CONTROL-BLOCKS=*XVT, -
                    MODULE=DOPEN) ——————————————————————————————————  (3)
PRINT-LIST OUTPUT=#REPORT ——————————————————————————————————————————  (4)
END
/MODIFY-JOB-SWITCHES OFF=5
/END-PROCEDURE
```

(1) The dump file to be analyzed is specified with `START-LIST-GENERATION`.

(2) The TCB and pages 0 to FFF are to be output for the task SRPM.

(3) The global XVT objects and the module DOPEN are to be output.

(4) Output is sent to the temporary file #REPORT.

Example 2

```
/BEGIN-PROCEDURE
/ASSIGN-SYSDTA TO-FILE=*SYSCMD
/ADD-FILE-LINK FILE=SYSDUMP.FROM.YESTERDAY, LINK=#1 ——————————————————  (1)
/MODIFY-JOB-SWITCHES ON=5
/START-DAMP
OPEN-DIAGNOSIS-OBJECT OBJECT=*#1 ———————————————————————————————————  (2)
START-LIST-GENERATION ——————————————————————————————————————————————  (3)
PRINT-LIST ———————————————————————————————————————————————————————————  (4)
END
/MODIFY-JOB-SWITCHES OFF=5
/END-PROCEDURE
```

(1) `/ADD-FILE-LINK` assigns the dump file to be evaluated.

(2) The `OPEN-DIAGNOSIS-OBJECT` statement opens the dump file for processing by DAMP.

(3) List output is started. A `FILE-TO-EVALUATE` assignment (as seen in example 1) can be omitted, since the file opened and the file to be output are one and the same.

(4) Output is directed to SYSLST.

DAMP selects the areas to be output (minimum output).

5.4.3 Components and scope of the output lists

The components and scope of the output lists are contingent upon:

– the dump from which they originate; the different dump forms, e.g. SLED dumps and area dumps, have different contents and scopes, and this is reflected in the lists
– the type of list selected (MINIMUM or ALL).

The following overview shows the different types of list and what they are made up of.

Overview of the scope of the minimum and full analysis

List type Components	Area	User Min	User All	Sys Min	Sys All	SLED Min	SLED All	SNAP
General dump file info	A	A	A	A	A	A	A	A
System overview	P	P	P	A	A	A	A	P
Trace manager	-	-	-	A	A	A	A	P
Descriptors	-	-	-	A	A	A	A	-
CSECT-MAP privileged	-	-	-	A	A	A	A	A
REPLOG	-	-	-	A	A	A	A	-
Trace table (total)	-	A	A	A	A	A	A	A
Class 1 memory	-	-	-	R	A	R	A	A
Class 2 memory	-	-	-	R	A	R	A	-
Class 3/4 memory	-	-	-	R	A	R	A	T
AUDIT tables (processor-local)	-	-	-	A	A	A	A	-
CSECT-MAP nonprivileged	-	A	A	A	A	-	-	-
Trace table (task)	-	A	A	A	A	E	E	E
TCB	A	A	A	A	A	E	E	E
TFT	-	D	D	D	D	D	D	-
PCB	A	A	A	A	A	E	E	E
SPL stacks	-	-	-	A	A	E	E	-
Audit tables	-	A	A	A	A	E	E	-
Class 6 memory	T	R	A	R	A	-	-	-
Class 5 memory	T	A	A	A	E	E	E	-

Key:
- \- is not included
- A is included
- P partial areas are included
- R only referenced pages are included
- D only for special DMS evaluation
- E data output once for each error task or explicitly selected task

Table 8: Components and scope of the output lists

Description of the individual sections of the list:

Output of the general dump file information

This section contains the output from the status window in INF mode and is of particular importance if the dump file contains a number of objects. The object currently selected is displayed.

The section is contained in each of the modes SYS, MIN and ALL under SELECT but can also be requested explicitly with SEL=INF.

Output of the system overview

Each SLED, SNAP or system dump list requested with SEL=MIN/ALL starts with a system overview which is identical to that requested explicitly via SEL=SYS.

The system overview shows

– which system environment existed at the time of the dump,
– which global system problems, if any, existed at this time and
– which further system files belong to the diagnostic environment.

The overview contains the following items of information:

– server type
– memory configuration
– number of active processors
– BS2000 file name
– loader name
– name of the active SERSLOG file
– name of the active CONSLOG file
– overview of the software configuration (subsystems)
– overview of the active traces managed by the trace manager
– the system parameters set
– references to global system problems (saturation, FORCE-JOB-CANCEL, etc.)
– any error descriptors created during automatic preanalysis
– for SLED lists, the entire system trace table, the processor-local linkage AUDIT tables and the task overview, together with the system type-dependent hardware areas
– for system dumps and possibly user or area dumps, the task-specific edited system trace table, the process control blocks (PCBs), and program manager stacks of the task, the TCB and the task-local AUDIT tables.

Output of the CSECT list

The CSECT list is output per subsystem and version for privileged and nonprivileged subsystems, the CSECTs being sorted by their addresses. The ETPND module information is also output.

In order to save paper, the CSECT list, sorted by names, contains only the associated addresses of the CSECTs; it is intended simply as a cross-reference list (name → address).

For SLED or SNAP, the list generated by default does not contain the module version numbers. These can be obtained by explicitly marking CS or ALL in the MAPS field in the LIST window, but it should be noted that the generation of such a list takes several minutes.

Output of REP information

The REP information contained in the dump (REPLOG or SAVEREP in earlier operating system versions) is output.

Output of the system trace table

The system trace table is output both in its entirety and also in the form of an extract containing all entries for the task concerned (once per task-specific analysis). Matching entries can be identified by reference to the trace entry numbers.

Output of the process control blocks (PCBs)

The PCBs are edited and output. The edited PCBs are followed by the PCB chain in non-edited format (dump format).

Output of the SPL stacks

The SPL stacks are output in edited form.

Output of the system tables

The XVT and the TCB are output without being edited (dump format); the addresses are relocated to the start of the table.

The JCB and JTBP are not output: only selected information from these tables (such as the name of the loaded program) is edited and output.

Output of the task file table and task-local DMS tables

The TFT and the task-local DMS tables are output only if the parameter PROC *DMS is selected or if the automatic preanalysis determines that DMS is the faulty component. Global DMS tables are not processed by DAMP.

Output of memory pages

The memory pages are output as follows:

– Minimum output (default value):
  The memory pages of memory classes 1, 2, 3, 4 and 6 are output only if they are referenced via a register of a PCB, an SPL stack or a save area.
  Any memory pages which were addressed during a preceding interactive diagnosis, implicitly during the automatic pre-diagnosis or during the initialization of DAMP are also regarded as marked.
  The class 5 memory is output fully, depending on the contents of the dump file.
  Pages marked as “secret pages” may or may not be contained in the dump file, depending on the system parameter DUMPSEPA.

– Full output:
  All pages of memory classes 1, 2, 3, 4, 5 and 6 which are included in the dump file are output.

5.5 Automating operations

5.5.1 Automatic preanalysis

PRODAMP procedures providing a basis for automatic preanalysis are supplied as standard components of DAMP. The diagnosis steps carried out during analysis and the resultant diagnostic information

– generate an internal list of referenced memory pages which are relevant for diagnosis,
– produce a description of the cause of the error and of the error environment, and
– attempt to pinpoint the error to a specific component (and, in the case of SLED and SNAP files, also to the error task).

The results of the preanalysis affect the way in which the dump is subsequently edited for printing: additional tables are output and the list is restricted to the error task.

Starting the preanalysis
– for a list analysis (`START-LIST-GENERATION)` (see page 158) or
– by calling a PRODAMP procedure with the statement `START-PRODAMP-PROGRAM NAME=DIAG`.

A SLED, SNAP or system dump must already be assigned with the `OPEN-DAIGNOSIS-OBJECT` statement.

The relevant PRODAMP routines are supplied in the file $TSOS.SYSDMP.DAMP. When you start the PRODAMP routine DIAG with `START-PRODAMP-PROGRAM`, you should ensure that this system PRODAMP library is assigned as the current user PRODAMP library. If necessary, you must assign the library with the `ASSIGN-PRODAMP-LIBRARIES` statement. You can display the current setting with the `SHOW-PRODAMP-LIBRARIES` statement.

Indexing, i.e. briefly describing the problem in hand in the form of a chain of descriptors (= descriptive keywords), is an established method for detecting duplicates. Automatic indexing of a problem with DAMP produces a string of such descriptors, which are output

– **on paper** at the beginning of the output list (as part of the system overview) if preanalysis is performed within the context of a list evaluation.

– **on the terminal** in EDT area 8, which can be accessed via the DAMP statement EDT and the EDT statement “[$]8”.

If preanalysis is called by means of `START-PRODAMP-PROGRAM NAME=DIAG`, no implicit print editing is carried out.

Automatic preanalysis also attempts to allocate the problem in hand to an error component. If this error component belongs to the subsystem DMS (allocator, catalog management, open/close, tape processing, access methods other than ISAM), the descriptor “DMS” is placed in the descriptor string.

Location of the error component in the subsystem DMS also affects print editing: task-specific DMS tables are output in addition to the system tables of the standard list.

In the case of SLED and SNAP files, automatic preanalysis controls the scope of list editing if an unambiguous error task is detected (e.g. $CRASH or a program error in a pre-allocated task). During standard editing, task-specific evaluation of this task is then initiated.

Preanalysis for task-specific problems in SLED or SNAP files (e.g. for the tasks which, at the time of the SLED run, were displaying the query "DUMP desired? YES/NO") can be controlled via task selection in the list mask (DIAG=YES/NO). If this is done, the descriptor "TASK.ONLY" is included in the string.

Automatic preanalysis cannot reveal all problems correctly, nor can it carry out in-depth analysis of the ones it does detect. Particularly in the case of SLED analyses and, within these, in the case of deadlock problems, the results of automatic preanalysis will often be of little practical use.

5.5.2 Batch and procedure modes, statement sequences

DAMP can also be started from within a BS2000 procedure or executed as a batch job. A procedure file can, for example, contain the necessary ADD-FILE-LINK commands for assigning the dump and logging files, together with the appropriate DAMP statements for assigning user-defined symbol files, for switching on the logging function or for calling dump analysis and print editing routines. Before the DAMP program is called, task switch 5 must be set in the procedure.

If, in procedure mode, a DAMP statement is followed by a system command that is not permitted at the DAMP program level (see page 224), DAMP switches from procedure mode to interactive mode and resets task switch 5. The last DAMP output screen is then displayed on the screen. In batch mode, a system command which is not permitted causes the job to be aborted.

Statement sequences which are required frequently during diagnosis with DAMP can also be stored in files. Such a file can then be activated in interactive, batch or procedure mode by means of the START-STATEMENT-SEQUENCE statement. In interactive mode, the last output which was initiated is displayed on the screen after the statement sequence has been executed. In procedure or batch mode, processing of the procedure or batch file is continued when all statements in the sequence have been executed.

When the START-STATEMENT-SEQUENCE statement is used, SYSDTA is temporarily redirected to the specified file. Changing the assignment for SYSDTA in this procedure (or, for example, in a procedure called via START-MODULE) may have unexpected consequences. In particular, the statement START-MODULE DCM should never be used in a statement file, since this can result in an endless loop.

Examples of statement sequences:

Contents of the file DAMP.STATEMENT.DCM

`LOG-SESSION`	Switch on logging
`OPEN-DIAGNOSIS-OBJECT OBJECT=#1`	Assign the dump file
`ADD-SYMBOLS MY.SYMBOLS(PCS(<ver>))`	Private symbol file PCS
`START-PATTERN-SEARCH`	Prepare string search
`SHOW-EDITED-INFO INFO=*TRACE, WINDOW=8`	Trace output to window W8
`MODIFY-SCREEN FIRST=3(10), SECOND=4(8)`	Window order for 1st screen

5.5.3 Automation with PRODAMP

PRODAMP (PROcedure language for DAMP) is a language similar to Pascal designed for the formulation of diagnostic algorithms in DAMP. With PRODAMP, it is possible to write decision-based statements, which would otherwise have to be entered individually by hand, into a procedure and to execute them automatically. It is possible, for example, to follow chains down to a structure which contains a specific data item, to search tables and process (e.g. arithmetically) the values they contain, or to have questions such as “Is this task holding a lock?” answered automatically.

PRODAMP is described in detail in section “PRODAMP” on page 227.

5.6 Program statements

5.6.1 Program level

DAMP statements must be entered in the command line on the screen. The SDF dialog interface supports interactive entry. The metasyntax for the applications is described in the “Commands” manual [8].
Since the command line can only accommodate 75 characters, longer statements must be abbreviated in such a way that there is no risk of confusion.
In batch and procedure mode, statements are the only type of entry permitted and are the only entries read and processed by SYSDTA.

The following DAMP statements are available. They are sorted according to fields of application:

Handling the diagnosis object

OPEN-DIAGNOSIS-OBJECT	Open diagnosis object for processing
ADD-SYMBOLS	Assign symbols for output
MODIFY-OBJECT-ASSUMPTIONS	Modify default settings for diagnosis object

Controlling representation

MODIFY-SCREEN-LAYOUT	Define new sequence and size for diagnostic windows
SHOW-EDITED-INFORMATION	Output specially edited diagnosis data
USE-REGISTER	Define representation for disassembled output
DROP-REGISTER	Cancel setting made with USE-REGISTER

Logging and repeating a DAMP session

LOG-SESSION	Activate logging of the diagnosis run. This creates a logging file.
REPEAT-SESSION	Replay a diagnostic log
PRINT-LOGGING-FILE	Edit and print a logging file.
STOP-LOGGING	Terminates logging of the diagnosis run started with LOG-SESSION.

Supporting automated diagnosis runs

a) PRODAMP

SHOW-PRODAMP-LIBRARIES	Show current PRODAMP libraries
ASSIGN-PRODAMP-LIBRARIES	Assign libraries for the PRODAMP compiler and/or PRODAMP editor
START-PRODAMP-EDITOR	Call editor for the PRODAMP-COMPILER
COMPILE-PRODAMP-PROCEDURE	Compile PRODAMP procedures outside of the PRODAMP editor
START-PRODAMP-PROGRAM	Load and start a PRODAMP program
RESUME-PRODAMP-PROGRAM	Resume an interrupted PRODAMP program

b) External subroutines via VMOS linkage

LOAD-MODULE	Load external subroutine
START-MODULE	Start external subroutine

c) DAMP procedures

START-STATEMENT-SEQUENCE	Read and execute DAMP statements from file

Creating lists

START-LIST-GENERATION	Initiate list output
ADD-LIST-OBJECTS	Specify scope of list by adding areas to be output
REMOVE-LIST-OBJECTS	Specify scope of list by excluding areas from output
PRINT-LIST	Start list output and specify output destination

Miscellaneous statements

START-PATTERN-SEARCH	Initiate string search
START-OPTION-DIALOG	Set user options
EDIT-FILE	Call EDT as a subroutine
SHOW-LAST-STATEMENT	Show last DAMP statement
END	Terminate DAMP
SEARCH-IN-SUBSYSTEM	Restrict CSECT search to specific subsystem
SHOW-SUBSYSTEM-FOR-SEARCH	Display subsystem selected for CSECT search

DAMP statements using the system command INFORM-PROGRAM

Operand MSG	Communicate with DAMP from system level

Details on controlling DAMP functions with the system command INFORM-PROGRAM can be found in section “System level” on page 224.

The following sections describe the DAMP statements in alphabetical order.

ADD-LIST-OBJECTS
Define scope of list output

The ADD-LIST-OBJECTS statement specifies the scope of list output. All instances of this statement which occur between START-LIST-GENERATION and PRINT-LIST are collected and taken into account when the PRINT-LIST statement is issued. Similarly, all instances of the REMOVE-LIST-OBJECTS statement are registered and taken into account.
Due to the complexity of the statement, it is advisable to use multiple ADD-LIST-OBJECTS statements when many different objects are to be selected.

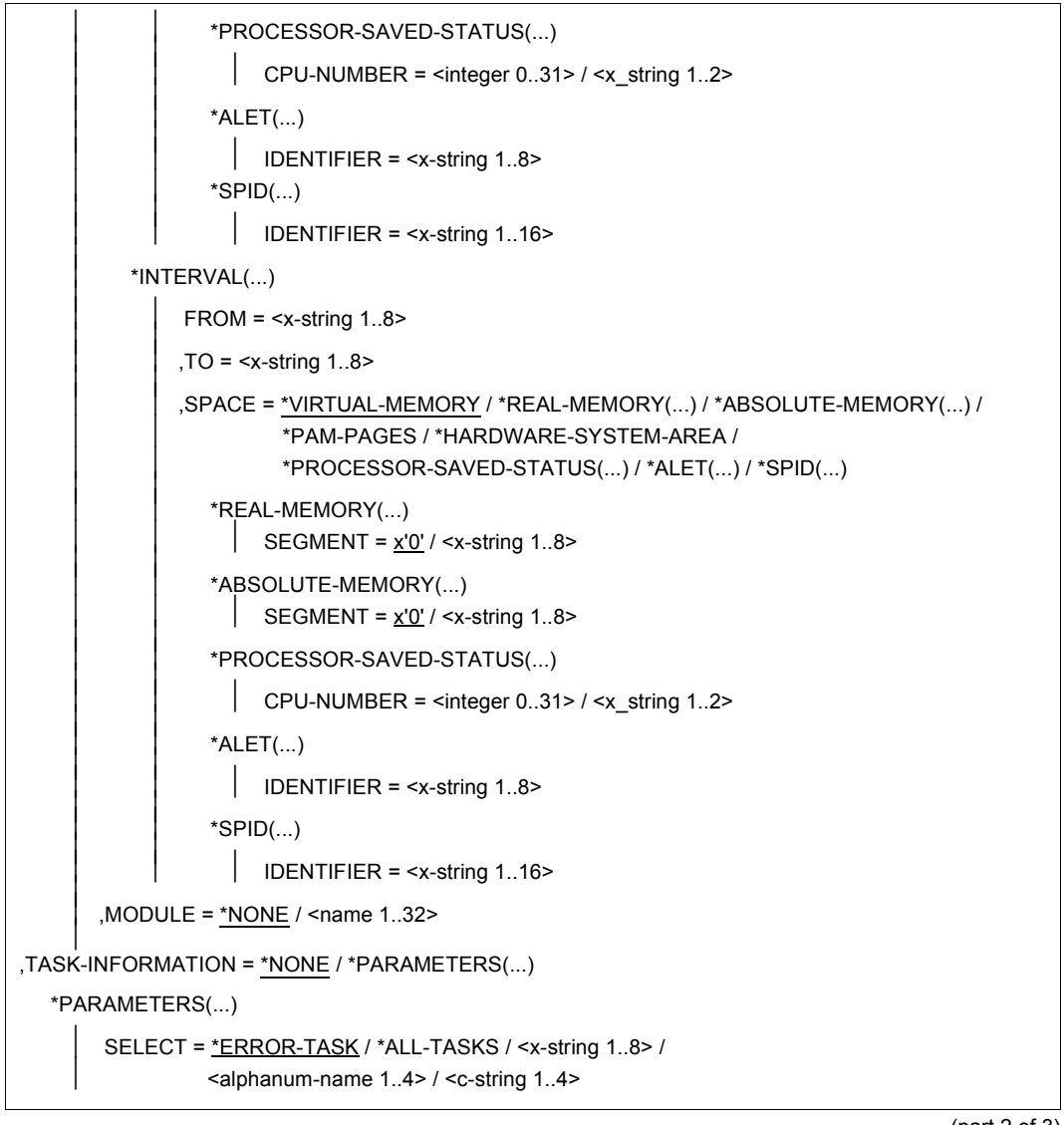
Format

```
ADD-LIST-OBJECTS

GLOBAL-INFORMATION = *NONE / *INF / *STD / *OVERVIEW / *ALL-MEMORY-AREAS /
                     *PARAMETERS(...)

  *PARAMETERS(...)
    | TRACES = *NONE / *ALL / list-poss(2): *SYSTEM-TRACE-TABLE / *TRACE-MANAGER-TABLES
    |
    | ,MAPS = *NONE / *ALL / list-poss(2): *CSECT-MAPS / *CONCISE-REP-INFORMATION
    |
    | ,CONTROL-BLOCKS = *NONE / *ALL / list-poss(2): *XVT / *AUDIT-TABLES
    |
    | ,MEMORY-AREAS = *NONE / *ALL / list-poss(9): *CL1 / *CL2 / *CL3 /
    |                   *CL3-PARTIAL-PAGES / *CL3-FULL-PAGES / *CL4 / *CL4-PARTIAL-PAGES /
    |                   *CL4-FULL-PAGES / *CL4-NON-PRIVILEGED
    |
    | ,PAGES = *NONE / <x-string 1..8>(...) / *INTERVAL(...)
    |
    |   <x-string 1..8>(...)
    |     |
    |     | SPACE = *VIRTUAL-MEMORY / *REAL-MEMORY (...) / *ABSOLUTE-MEMORY (...) /
    |     |           *PAM-PAGES / *HARDWARE-SYSTEM-AREA /
    |     |           *PROCESSOR-SAVED-STATUS(...) / *ALET(...) / *SPID(...)
    |     |
    |     |   *REAL-MEMORY(...)
    |     |     |
    |     |     | SEGMENT = x'0' / <x-string 1..8>
    |     |
    |     |   *ABSOLUTE-MEMORY(...)
    |     |     |
    |     |     | SEGMENT = x'0' / <x-string 1..8>
```

(part 1 of 3)


*PROCESSOR-SAVED-STATUS(...)
CPU-NUMBER = <integer 0..31> / <x_string 1..2>
*ALET(...)
IDENTIFIER = <x-string 1..8>
*SPID(...)
IDENTIFIER = <x-string 1..16>
*INTERVAL(...)
FROM = <x-string 1..8>
,TO = <x-string 1..8>
,SPACE = *VIRTUAL-MEMORY / *REAL-MEMORY(...) / *ABSOLUTE-MEMORY(...) /
*PAM-PAGES / *HARDWARE-SYSTEM-AREA /
*PROCESSOR-SAVED-STATUS(...) / *ALET(...) / *SPID(...)
*REAL-MEMORY(...)
SEGMENT = x'0' / <x-string 1..8>
*ABSOLUTE-MEMORY(...)
SEGMENT = x'0' / <x-string 1..8>
*PROCESSOR-SAVED-STATUS(...)
CPU-NUMBER = <integer 0..31> / <x_string 1..2>
*ALET(...)
IDENTIFIER = <x-string 1..8>
*SPID(...)
IDENTIFIER = <x-string 1..16>
,MODULE = *NONE / <name 1..32>
,TASK-INFORMATION = *NONE / *PARAMETERS(...)
*PARAMETERS(...)
SELECT = *ERROR-TASK / *ALL-TASKS / <x-string 1..8> /
<alphanum-name 1..4> / <c-string 1..4>

(part 2 of 3)

```
,INFORMATION = *STD / *INF / *OVERVIEW / *ALL-MEMORY-AREAS / *PARAMETERS(...)
    *PARAMETERS(...)
        TRACES = *NONE / *SYSTEM-TRACE-TABLE
        ,MAPS = *NONE / *USER-CSECTS
        ,CONTROL-BLOCKS = *NONE / *ALL / list-poss(5): *TCB / *PCBS / *SPL-STACKS /
                          *TFTS / *AUDIT-TABLES
        ,MEMORY-AREAS = *NONE / *ALL / list-poss(7): *CL5 / *CL5-PARTIAL-PAGES /
                        *CL5-FULL-PAGES / *CL5-MEMORY-POOLS / *CL6 /
                        *CL6-FULL-PAGES / *CL6-MEMORY-POOLS
        ,PAGES = *NONE / <x-string 1..8>(...) / *INTERVAL(...)
            <x-string 1..8>>(...)
                SPACE = *VIRTUAL-MEMORY / *ALET(...)
                    *ALET(...)
                        IDENTIFIER = <x-string 1..8>
            *INTERVAL(...)
                FROM = <x-string 1..8>
                ,TO = <x-string 1..8>
                ,SPACE = *VIRTUAL-MEMORY / *ALET(...)
                    *ALET(...)
                        ,IDENTIFIER = <x-string 1..8>
        ,MODULE = *NONE / <name 1..32>
,USER-LIST-PROCEDURE = *NONE / <name 1..32 with-under> / <structured-name 1..32>
,WINDOW = *NONE / <integer 4..99>
,DUMP-DIAGNOSIS = *NONE / *PREANALYSIS(...)
    *PREANALYSIS(...)
        ERROR-DESCRIPTION = *NO / *YES
```

(part 3 of 3)

Operands

GLOBAL-INFORMATION = *NONE / *INF / *STD / *OVERVIEW / *ALL-MEMORY-AREAS / *PARAMETERS(...)
This operand is used to select global (as opposed to task-specific) areas for list output.

GLOBAL-INFORMATION = *NONE
The list output is not to include any global memory areas.

GLOBAL-INFORMATION = *INF
The list output should contain general information relating to the object to be analyzed.

GLOBAL-INFORMATION = *STD
A minimal scope list is output. This list will generally contain all the data required for preliminary diagnosis (see section “Components and scope of the output lists” on page 160).

GLOBAL-INFORMATION = *OVERVIEW
A system overview is output (see section “Components and scope of the output lists” on page 160 for the contents of this overview).

GLOBAL-INFORMATION = *ALL-MEMORY-AREAS
A complete list is output (see the section “Components and scope of the output lists” on page 160 for the contents of this list). In particular, all the pages saved by CDUMP are output to the list in the case of system, user and area dumps.

GLOBAL-INFORMATION = *PARAMETERS(...)
This operand allows global areas to be selected individually.

TRACES = *NONE / *ALL / list-poss(2): *SYSTEM-TRACE-TABLE / *TRACE-MANAGER-TABLES
This operand specifies the system traces to be output.

TRACES = *NONE
No system traces are to be output. This is the default value.

TRACES = *ALL
All the system traces are to be output.

TRACES = list-poss(2): *SYSTEM-TRACE-TABLE / *TRACE-MANAGER-TABLES
The system trace table and an overview of the traces managed by the trace manager can be selected individually or in the form of a list.

MAPS = *NONE / *ALL / list-poss(2): *CSECT-MAPS / *CONCISE-REP-INFORMATION
This parameter specifies the information to be output on loaded CSECTs and on the correction status of the system.

MAPS = *NONE
No maps are to be output.

MAPS = *ALL
Both CSECT maps and the REPLOG are to be output if they exist in the diagnosis object.

MAPS = list-poss(2): *CSECT-MAPS / *CONCISE-REP-INFORMATION
The CSECT maps and the REPLOG can be selected individually or in the form of a list.

CONTROL-BLOCKS = *NONE / *ALL / list-poss(2): *XVT / *AUDIT-TABLES
This operand specifies which tables are to be output.

CONTROL-BLOCKS = *NONE
No tables are to be output.

CONTROL-BLOCKS = *ALL
The XVT and AUDIT tables are to be output.

CONTROL-BLOCKS = list-poss(2): *XVT / *AUDIT-TABLES
The global system table XVT and the AUDIT tables can be selected individually or in the form of a list.
If the XVT is selected, it is output in unedited form. In the case of the AUDIT tables, any processor-local linkage AUDIT tables are to be output.

MEMORY-AREAS = *NONE / *ALL / list-poss(9): *CL1 / ... / *CL4-NON-PRIVILEGED
This parameter specifies certain global areas of the virtual address space which are identified by their memory class and which are to be output in their entirety in standard dump format. The default value specifies that none of these areas are to be output. The operand *ALL combines all the memory areas listed.

CL1:	resident system modules
CL2:	pageable system modules
CL3:	resident tables and subsystem modules
CL3-PARTIAL-PAGES:	resident partial pages
CL3-FULL-PAGES:	resident full pages
CL4:	pageable tables and subsystem modules
CL4-PARTIAL-PAGES:	pageable partial pages
CL4-FULL-PAGES:	pageable full pages
CL4-NON-PRIVILEGED:	nonprivileged memory system pages

PAGES = *NONE / <x-string 1..8>(...) / *INTERVAL(...)
Direct selection of memory pages or memory areas in the address space. The default setting does not specify any particular pages for output.

PAGES = <x-string 1..8>(...)
Direct selection of a memory page.

SPACE = *VIRTUAL-MEMORY / *REAL-MEMORY /
***ABSOLUTE_MEMORY/*PAM-PAGES/*HARDWARE-SYSTEM-AREA /**
***PROCESSOR-SAVED-STATUS(...) / *ALET(...) / *SPID(...)**
This operand specifies the memory type referred to by the page specification.

SPACE = *VIRTUAL-MEMORY
If no specification is made, VIRTUAL-MEMORY is assumed, i.e. virtual memory is used by default.

SPACE = *REAL-MEMORY(...)
This operand indicates that the main memory is to be used (real address space).

SEGMENT = x'0' / <x-string 1..8>
Indicates the 4 GB segment (0,1,...) associated with the real page.
Segment 0 is the default.

SPACE = *ABSOLUTE-MEMORY(...)
This operand indicates the host-absolute addresses (e.g. in the case of a complete VM2000 SLED file).

SEGMENT = x'0' / <x-string 1..8>
Indicates the 4 GB segment (0,1,...) associated with the absolute page.
Segment 0 is the default.

SPACE = *PAM-PAGES
The page specification refers to PAM pages.

SPACE = *HARDWARE-SYSTEM-AREA
This operand indicates that the hardware system area HSA is to be used (this area contains, for instance, tables used for communication between the CPU and the I/O processor). The specified page must lie within the HSA.

SPACE = *PROCESSOR-SAVED-STATUS(...)
The page specification refers to a processor saved status area.

CPU-NUMBER = <integer 0..31>
If page 0 of the processor saved status area is to be output, the CPU number must be specified in decimal format.

CPU-NUMBER = <x-string 1..2>
If page 0 of the processor saved status area is to be output, the CPU number must be specified in hexadecimal format.

SPACE = *ALET(...)
This operand indicates that pages from data spaces are to be used (data spaces are extensions of the system's virtual address space). The ALET (access list entry token) is used to identify the data space for the system address space.

IDENTIFIER = <x-string 1..8>
The 4-byte ALET is specified in hexadecimal format.

SPACE = *SPID(...)
This operand indicates that pages from data spaces are to be used (data spaces are extensions of the system's virtual address space). The SPID (space identification) is used to identify the data space throughout the system.

IDENTIFIER = <x-string 1..16>
The 8-byte SPID is specified in hexadecimal format.

PAGES = *INTERVAL(...)
This parameter allows a number of memory pages to be specified by means of an interval.

FROM = <x-string 1..8>
This operand indicates the first memory page of the memory area.

TO = <x-string 1..8>
This operand indicates the last memory page of the memory area.

SPACE = *VIRTUAL-MEMORY / *REAL-MEMORY(...) / *ABSOLUTE-MEMORY(...) / *PAM-PAGES / *HARDWARE-SYSTEM-AREA / *PROCESSOR-SAVED-STATUS(...) / *ALET(...) / *SPID(...)
This operand specifies the type of memory to which the page specification refers.

SPACE = *VIRTUAL-MEMORY
If no specification is made, VIRTUAL-MEMORY is assumed, i.e. virtual memory is used by default.

SPACE = *REAL-MEMORY(...)
This operand indicates that the main memory is to be used (real address space).

SEGMENT = x'0' / <x-string 1..8>
Indicates the 4 GB segment (0,1,...) associated with the real page.
Segment 0 is the default.

SPACE = *ABSOLUTE-MEMORY(...)
This operand indicates the host-absolute addresses (e.g. in the case of a complete VM2000 SLED file).

SEGMENT = x'0' / <x-string 1..8>
Indicates the 4 GB segment (0,1,...) associated with the absolute page.
Segment 0 is the default.

SPACE = *PAM-PAGES
The page specification refers to PAM pages.

SPACE = *HARDWARE-SYSTEM-AREA
This operand indicates that the hardware system area HSA is to be used. The specified pages must lie within the HSA.

SPACE = *PROCESSOR-SAVED-STATUS(...)
The page specification refers to a processor saved status area.

CPU-NUMBER = <integer 0..31>
If page 0 of the processor saved status area is to be output, the CPU number must be specified in decimal format.

CPU-NUMBER = <x-string 1..2>
If page 0 of the processor saved status area is to be output, the CPU number must be specified in hexadecimal format.

SPACE = *ALET(...)
This operand indicates that pages from data spaces are to be used (data spaces are extensions of the system's virtual address space). The ALET (access list entry token) is used to identify the data space for the system address space.

IDENTIFIER = <x-string 1..8>
The 4-byte ALET is specified in hexadecimal format.

SPACE = *SPID(...)
This operand indicates that pages from data spaces are to be used (data spaces are extensions of the system's virtual address space). The SPID (space identification) is used to identify the data space throughout the system.

IDENTIFIER = <x-string 1..16>
The 8-byte SPID is specified in hexadecimal format.

MODULE = *NONE / <name 1..32>
This operand specifies the entire memory area occupied by the specified system module. The default setting specifies that this area is not output.

TASK-INFORMATION = *NONE / *PARAMETERS(...)
This operand is used to select task-specific areas.

TASK-INFORMATION = *NONE
The list output is to contain no task-specific memory areas.

TASK-INFORMATION = *PARAMETERS(...)
Selection of specified task-specific memory areas.

SELECT = *ERROR-TASK / *ALL-TASKS / <x-string 1..8> / <alphanum-name 1..4> / <c-string 1..4>
Selection of the task for which memory areas are to be output.

SELECT = *ERROR-TASK
Selection of the error task. For system, user and area dumps, this is the only task contained in the dump file. With SLEDs, the error task is defined by the automatic preanalysis within DAMP.

SELECT = *ALL-TASKS
In the case of a SLED, task editing is carried out for all active tasks.

SELECT = <x-string 1..8>
The required task is specified by the 4-byte TID in hexadecimal format.

SELECT = <alphanum-name 1..4>
The required task is identified by an alphanumeric name which is interpreted as the task's TSN.

SELECT = <c-string 1..4>
The required task is identified by a character string which is interpreted as the task's TSN.

INFORMATION = *STD / *INF / *OVERVIEW / *ALL-MEMORY-AREAS / *PARAMETERS(...)
This operand controls the scope of output for task-specific data.

INFORMATION = *STD
A minimal scope list is output. This list will generally contain all the data required for preliminary diagnosis (see section "Components and scope of the output lists" on page 160).

INFORMATION = *INF
The output list contains general information on the selected task.

INFORMATION = *OVERVIEW
A system overview is output (see section "Components and scope of the output lists" on page 160 for the contents of this overview).

INFORMATION = *ALL-MEMORY-AREAS
A complete list is output (see section "Components and scope of the output lists" on page 160 for the contents of this list). In particular, all the pages saved by CDUMP are output to the list in the case of system, user and area dumps.

INFORMATION = *PARAMETERS(...)
This operand allows task-specific areas to be selected individually.

TRACES = *NONE / *SYSTEM-TRACE-TABLE
This operand defines whether task-specific information is to be output from the system trace table. Default: no output.

MAPS = *NONE / *USER-CSECTS
This operand controls the output of information on the loaded user CSECTs. Information on user CSECTs can only be output for user and area dumps if the binder/loader information in class 5 memory is contained in the object.
Default: no output.

CONTROL-BLOCKS = *NONE / *ALL / list-poss(5): *TCB / ... / *AUDIT-TABLES
This operand specifies whether the task-specific control blocks are to be output in standard dump format.

CONTROL-BLOCKS = *NONE
The task-specific control blocks are not output. This is the default setting.

CONTROL-BLOCKS = *ALL
All task-specific control blocks are to be output.

CONTROL-BLOCKS = list-poss(5): *TCB / *PCBS / *SPL-STACKS / *TFTS / *AUDIT-TABLES
This operand allows explicit selection of task-specific control blocks. The blocks can be selected individually or in the form of a list. TCB (task control block), PCBS (process control blocks), SPL-STACKS, (TPR program manager stacks), TFTS (task file tables), and AUDIT-TABLES (hardware AUDIT and linkage AUDIT).
If PCBS is specified, the PCBs are additionally output in edited format.

MEMORY-AREAS = *NONE / *ALL / list-poss(7): *CL5 / ... / *CL6-MEMORY-POOLS
This parameter specifies certain areas of the virtual task address space which are identified by their memory class and which are to be output in their entirety in standard dump format.

MEMORY-AREAS = *NONE
No special areas of the virtual task address space are selected.

MEMORY-AREAS = *ALL
All memory areas of class 5 and class 6 memory are to be output in standard dump format.

MEMORY-AREAS = list-poss(7): *CL5 / *CL5-PARTIAL-PAGES / *CL5-FULL-PAGES / *CL5-MEMORY-POOLS / *CL6 / *CL6-FULL-PAGES / *CL6-MEMORY-POOLS
The class 5 and class 6 memory segments can be selected individually or in the form of a list.

CL5:	complete class 5 memory
CL5-PARTIAL-PAGES:	privileged partial pages of class 5 memory
CL5-FULL-PAGES:	privileged full pages of class 5 memory
CL5-MEMORY-POOLS:	memory pool pages of class 5 memory
CL6:	complete class 6 memory
CL6-FULL-PAGES:	nonprivileged full pages of class 6 memory
CL6-MEMORY-POOLS:	memory pool pages of class 6 memory

PAGES = *NONE / <x-string 1..8>(...) / *INTERVAL(...)
Direct selection of memory pages to be output.

PAGES = *NONE
If no specification is made, *NONE is assumed, i.e. no specific memory pages are output.

PAGES = <x-string 1..8>(...)
This operand directly selects one specific memory page via a hexadecimal page specification.

SPACE = *VIRTUAL-MEMORY / *ALET(...)
This operand specifies the memory type referred to by the page specification.

SPACE = *VIRTUAL-MEMORY
If no specification is made, VIRTUAL-MEMORY is assumed, i.e. virtual memory is used by default.

SPACE = *ALET(...)
This operand indicates that pages from data spaces are to be used (data spaces are extensions of the system's virtual address space). The ALET (access list entry token) is used to identify the data space for the task address space.

IDENTIFIER = <x-string 1..8>
The 4-byte ALET is specified in hexadecimal format.

PAGES = *INTERVAL(...)
This parameter allows a number of memory pages to be specified by means of an interval.

FROM = <x-string 1..8>
This operand indicates the first memory page of the memory area.

TO = <x-string 1..8>
This operand indicates the last memory page of the memory area.

SPACE = *VIRTUAL-MEMORY / *ALET(...)
This operand specifies the memory type referred to by the page specification.

SPACE = *VIRTUAL-MEMORY
If no specification is made, VIRTUAL-MEMORY is assumed, i.e. virtual memory is used by default.

SPACE = *ALET(...)
This operand indicates that pages from data spaces are to be used (data spaces are extensions of the system's virtual address space). The ALET (access list entry token) is used to identify the data space for the task address space.

IDENTIFIER = <x-string 1..8>
The 4-byte ALET is specified in hexadecimal format.

MODULE = *NONE / <name 1..32>
This operand specifies the name of a module from the loaded user program, which is to be output in its entirety in standard dump format. If no specification is made, *NONE is assumed, i.e. no module is assigned.

USER-LIST-PROCEDURE = *NONE / <name 1..32 with under> / <structured-name 1 .. 32>
This operand specifies the name of a PRODAMP program located in the currently set user PRODAMP object library. This program is started automatically once output of this list has been completed.

WINDOW = *NONE / <integer 4..99>
The specified window is output to a list using the current layout. Windows 4 - 9 and 21 - 99 are supported; windows 10 - 20 are 'reserved'. Only windows using standard format (dump format, hexadecimal format, character format, assembled or symbolic) can be output.

DUMP-DIAGNOSIS = *NONE / *PREANALYSIS(...)
This operand indicates whether automatic preanalysis of the diagnostic data is to be performed.
It is equivalent to entering DIAG=YES/NO in the LIST window, i.e. the preanalysis procedure DIAG is either started or is not started.
The results of analysis regarding diagnosis-relevant memory pages, error causes, error environment and error localization influences the scope of the subsequent lists.

DUMP-DIAGNOSIS = *NONE
No preanalysis is to be performed.

DUMP-DIAGNOSIS = *PREANALYSIS(...)
The preanalysis procedure DIAG is to be started. DIAG is a PRODAMP procedure from the system PRODAMP library.
The automatic preanalysis of user and area dumps is currently not supported.

ERROR-DESCRIPTION = *NO / *YES
Indicates whether the error descriptors are to be output in the format of the retrieval system SIS at the beginning of the output list for the system overview. These descriptors can then be used to search for duplicates with SIS.

ERROR-DESCRIPTION = *NO
No error descriptors are to be output. However, the memory pages referenced during preanalysis and relevant to diagnosis are automatically included in output for any subsequent minimum evaluation.

ERROR-DESCRIPTION = *YES
Error descriptors are to be output.
This is equivalent to entering DESCR=YES in the LIST window, i.e. the preanalysis procedure DESCR is started. DESCR is a PRODAMP procedure from the system PRODAMP library.

Example

```
START-LIST-GENERATION
ADD-LIST GLOBAL-INFORMATION=*PAR(CONTROL-BLOCKS=*XVT) ——————————————————  (1)
ADD-LIST GLOBAL-INFORMATION=*PAR(MEMORY-AREAS=(*CL1,*CL3)) ———————————————  (2)
ADD-LIST TASK-INFORMATION=*PAR(X'00040333,
         INFO=(CONTROL-BLOCKS=*TCB,PAGES=X'14'(*ALET=X'05')))) ——————————  (3)
ADD-LIST TASK-INFORMATION=*PAR(3UVW,INFO=*PAR(MEM=*CL6)) ————————————————  (4)

PRINT-LIST
```

(1)/(2) Output of global system information: XVT, class 1 and class 3 memory.

(3) The TCB and page 14 from the data space with ALET X'00000005' is to be output for the task with TID X'00040333'.

(4) Class 6 memory is to be output for the task with TSN 3UVW.

See the PRINT-LIST statement on page 201.

ADD-SYMBOLS
Assign symbols for output

The ADD-SYMBOLS statement assigns additional symbol elements for use in editing symbolic output.

Format

```
ADD-SYMBOLS

LIBRARY = *STD(...) / <filename 1..54 without-gen-vers>(...)
    *STD(...)
        ELEMENT = <filename 1..54 without-cat-user-gen-vers>(...)
            <filename>(...)
                VERSION = <composed-name_1..24_with-under>
    <filename 1..54 without-gen-vers>(...)
        ELEMENT = <filename 1..54 without-cat-user-gen-vers>(...)
            <filename>(...)
                VERSION = <composed-name_1..24_with-under>
```

Operands

LIBRARY = *STD(...) / <filename 1..54 without-gen-vers>(...)
Specifies the name of the symbol library containing the symbol element.

LIBRARY = *STD(...)
The *STD operand value stands for the DAMP standard symbol library.

ELEMENT = <filename 1..54 without-cat-user-gen-vers>(...)
Specifies the name of the element containing the symbols. The symbol elements supplied as standard with DAMP have the same name as the product to which they belong, e.g. BS2000 or DAB.

VERSION = <composed-name_1..24_with-under>
Specifies the version ID of the symbol element. The symbol elements supplied as standard with DAMP have the same version ID as the version of the product to which they belong, e.g. 200 oder 200.x1, where x=A,B, ...

LIBRARY = <filename 1..54 without-gen-vers>(...)
Name of the symbol library.

ELEMENT = <filename 1..54 without-cat-user-gen-vers>(...)
Specifies the name of the element containing the symbols. The symbol elements supplied as standard with DAMP have the same name as the product to which they belong, e.g. BS2000 or DAB.

VERSION = <composed-name_1..24_with-under>
Specifies the version ID of the symbol element. The symbol elements supplied as standard with DAMP have the same version ID as the version of the product to which they belong, e.g. 200 or 200.x1, where x=A,B, ...

Examples

```
ADD-SYMBOLS SYSSMB.DAMP.<ver>(BS2000(200.H1))

ADD-S *STD(BS2000(200))
```

ASSIGN-PRODAMP-LIBRARIES
Assign libraries for PRODAMP compiler and PRODAMP editor

The ASSIGN-PRODAMP-LIBRARIES statement is used to assign one library each for PRODAMP source files and PRODAMP objects.

Format

```
ASSIGN-PRODAMP-LIBRARIES

 SOURCE-LIBRARY = *UNCHANGED / *STD / <filename 1..54 without-gen-vers> /
                  *PRODAMP-SYSTEM-LIBRARY
,OBJECT-LIBRARY = *UNCHANGED / *STD / *SOURCE-LIBRARY / <filename 1..54 without-gen-vers> /
                  *PRODAMP-SYSTEM-LIBRARY
```

Operands

SOURCE-LIBRARY = *UNCHANGED / *STD / <filename 1..54 without-gen-vers> / *PRODAMP-SYSTEM-LIBRARY
Specifies the name of the library the PRODAMP editor is to use when locating and storing PRODAMP source files.

SOURCE-LIBRARY = *UNCHANGED
The current name of the library is not changed.

SOURCE-LIBRARY = *STD
The user library defined in the OPTION window for which the standard SYS.USRDMP.DAMP.<ver> is specified is selected.

SOURCE-LIBRARY = <filename 1..54 without-gen-vers>
The specified library name is used for the PRODAMP sources.

SOURCE-LIBRARY = *PRODAMP-SYSTEM-LIBRARY
Designates the PRODAMP system library. For standard installations, this is the library $TSOS.SYSDMP.DAMP.

OBJECT-LIBRARY = *UNCHANGED / *STD / *SOURCE-LIBRARY / <filename 1..54 without-gen-vers> / *PRODAMP-SYSTEM-LIBRARY
Specifies the name of the library to be used by the PRODAMP compiler for storing PRODAMP objects and by the PRODAMP runtime system when loading PRODAMP objects.

OBJECT-LIBRARY = *UNCHANGED
The current name of the library is not changed.

OBJECT-LIBRARY = *STD
The user library defined in the OPTION window for which the standard SYS.USRDMP.DAMP.<ver> is specified is selected.

OBJECT-LIBRARY = *SOURCE-LIBRARY
Source library and object library are identical.

OBJECT-LIBRARY = <filename 1..54 without-gen-vers>
The specified library name is used for the PRODAMP objects.

OBJECT-LIBRARY = PRODAMP-SYSTEM-LIBRARY
Designates the PRODAMP system library. For standard installations, this is the library $TSOS.SYSDMP.DAMP.

Examples

```
ASSIGN-PRODAMP-LIBRARIES  SOURCE=LIB.FOR.PRODAMP, OBJECT=*SOURCE

ASSIGN  MY-SOURCE-LIB, MY-OBJECT-LIB
```

COMPILE-PRODAMP-PROCEDURE
Compile PRODAMP procedure

The statement COMPILE-PRODAMP-PROCEDURE is used to compile a PRODAMP procedure from a PRODAMP library (i.e. outside of the PRODAMP editor) in interactive mode or a batch job.

Format

```
COMPILE-PRODAMP-PROCEDURE

 SOURCE = *PRODAMP-USER-SOURCE-LIBRARY(...) / *PRODAMP-SYSTEM-LIBRARY(...) /
                <filename 1..54 without-gen-vers>(...)

    *PRODAMP-USER-SOURCE-LIBRARY(...)

        ELEMENT = <name 1..32 with-under>
        VERSION = *HIGHEST-EXISTING / <integer 1..999>

    *PRODAMP-SYSTEM-LIBRARY(...)

        ELEMENT = <name 1..32 with-under>
        VERSION = *HIGHEST-EXISTING / <integer 1..999>

    <filename 1..54 without-gen-vers>(...)

        ELEMENT = <name 1..32 with-under>
        VERSION = *HIGHEST-EXISTING / <integer 1..999>

,OBJECT-LIBRARY = *PRODAMP-USER-OBJECT-LIBRARY / *PRODAMP-SYSTEM-LIBRARY /
                      *SOURCE-LIBRARY / <filename 1..54 without-gen-vers>
```

Operands

SOURCE = *PRODAMP-USER-SOURCE-LIBRARY(...) / *PRODAMP-SYSTEM-LIBRARY(...) / <filename 1..54 without-gen-vers>(...)
Specifies the PRODAMP library from which the PRODAMP procedure is to be taken.

SOURCE = *PRODAMP-USER-SOURCE-LIBRARY(...)
The currently set PRODAMP user source library is used.

SOURCE = *PRODAMP-SYSTEM-LIBRARY(...)
The PRODAMP system library is used.
For standard installations this is the library $TSOS.SYSDMP.DAMP.

SOURCE = <filename 1..54 without-gen-vers>(...)
The library with the specified name is used.

ELEMENT = <name 1..32 with-under>
Name of the element to be compiled.

VERSION = *HIGHEST-EXISTING / <integer 1..999>
Version of the element to be compiled.

OBJECT-LIBRARY = *PRODAMP-USER-OBJECT-LIBRARY / *PRODAMP-SYSTEM-LIBRARY / *SOURCE-LIBRARY / <filename 1..54 without-gen-vers>
Specifies the PRODAMP library to which the PRODAMP program is to be saved.

OBJECT-LIBRARY = *PRODAMP-USER-OBJECT-LIBRARY
The PRODAMP program will be saved in the currently set PRODAMP user object library.

OBJECT-LIBRARY = *PRODAMP-SYSTEM-LIBRARY
The PRODAMP program will be saved in the PRODAMP system library.
For standard installations this is the library $TSOS.SYSDMP.DAMP.

OBJECT-LIBRARY = *SOURCE-LIBRARY
The PRODAMP program will be saved in the specified source library.

OBJECT-LIBRARY = <filename 1..54 without-gen-vers>
The PRODAMP program will be saved in the library with the specified name.

Example

```
COMPILE-PRODAMP-PROCEDURE SOURCE=MYLIB(ELEMENT=MYPROG),OBJECT-LIBRARY=*SOURCE
```

DROP-REGISTER
Define representation for disassembler

The DROP-REGISTER statement is used to cancel register declarations made by means of the USE-REGISTER statement. If the specified module is represented in disassembled format, the Assembler commands which use the specified register are edited using the base address and the offset (the instruction addresses are not shown in symbolic representation).

The DROP-REGISTER statement and the corresponding USE-REGISTER statement do not support the output of disassembled x86 code. x86 registers may not be specified.

Format

DROP-REGISTER
MODULE-NAME = <name 1..32> ,REGISTER = *ALL / <integer 0..15>

Operands

MODULE-NAME = <name 1..32>
Specifies the module or control block (DSECT) for which the register declarations are to be canceled.

REGISTER = *ALL / <integer 0..15>
Specifies one or more registers for which base address and offset are to be used for disassembled representation. If *ALL is specified, the format used for all registers previously specified in a USE-REGISTER statement is the base address and the offset format.

REGISTER = <integer 0..15>
Specifies a /390 general register.

Examples

```
DROP-REGISTER MODULE-NAME=DOPEN, REGISTER=12

DROP-REGISTER  DOPEN, *ALL
```

EDIT-FILE
Load EDT as subroutine

The EDIT-FILE statement calls EDT as a subroutine and loads any file specified into the EDT work area.

Format

EDIT-FILE
NAME = *NONE / <filename 1..54 without-gen-vers>

Operands

NAME = *NONE / <filename 1..54 without-gen-vers>
Specifies the name of a file to be read into the EDT work area. If the work area already contains a file, EDT is called but the new work file is not loaded. If this happens, an appropriate message is issued in the EDT message line.

Example

```
EDIT-FILE FILE.HUGO
```

END
Terminate DAMP

The END statement terminates DAMP.

This statement has no operands.

LOAD-MODULE
Load module from library

The LOAD-MODULE statement loads a module from a library. In order to do this, it must be possible to call the module via the VMOS linkage. The module can then be executed as often as required using the START-MODULE statement. This means that DAMP users can write their own external procedures and run them under DAMP in the way described.
A module loaded using LOAD-MODULE is only unloaded once DAMP is terminated.
LOAD-MODULE can be used within PRODAMP to improve the performance and increase the flexibility of ENTER_MODULE (see page 270).

Format

```
LOAD-MODULE

LIBRARY = *STD(...) / *PRODAMP-USER-OBJECT-LIBRARY(...) / *PRODAMP-SYSTEM-LIBRARY(...) /
          <filename 1..54 without-gen-vers>(...)

  *STD(...)
    | ELEMENT = <name 1..8>

  *PRODAMP-USER-OBJECT-LIBRARY(...)
    | ELEMENT = <name 1..8>

  *PRODAMP-SYSTEM-LIBRARY(...)
    | ELEMENT = <name 1..8>

  <filename 1..54 without-gen-vers>(...)
    | ELEMENT = <name 1..8>
```

Operands

LIBRARY = *STD(...) / *PRODAMP-USER-OBJECT-LIBRARY(...) / *PRODAMP-SYSTEM-LIBRARY(...) / <filename 1..54 without-gen-vers>(...)
Specifies the name of the library from which the module is to be loaded.

LIBRARY = *STD(...)
The DAMP module library is assigned.

ELEMENT = <name 1..8>
Name of the module to be loaded.

LIBRARY = *PRODAMP-USER-OBJECT-LIBRARY(...)
The module is loaded from the current user PRODAMP library.

ELEMENT = <name 1..8>
Name of the module to be loaded.

LIBRARY = *PRODAMP-SYSTEM-LIBRARY(...)
The module is loaded from the current system PRODAMP library. For standard installations, this is the library $TSOS.SYSDMP.DAMP.

ELEMENT = <name 1..8>
Name of the module to be loaded.

LIBRARY = <filename 1..54 without-gen-vers>(...)
Name of the library.

ELEMENT = <name 1..8>
Name of the module to be loaded.

Examples

```
LOAD-MODULE *STD(MYOWNUTI)

LOAD-MODULE ELEM=MYOWNUTI
```

LOG-SESSION
Activate logging of diagnosis run

The LOG-SESSION statement causes all subsequent inputs and outputs of the DAMP session to be logged in the specified file.

Format

LOG-SESSION
LOGGING-FILE = *STD / <filename 1..54>

Operands

LOGGING-FILE = *STD / <filename 1..54>
Name of the file in which the inputs and outputs are to be logged. If *STD is specified, DAMP generates the file name S.LOG.DAMP.<ver>.<date>.<time>.

Example

```
LOG-SESSION LOGGING-FILE=LOG.HUGO
```

MODIFY-OBJECT-ASSUMPTIONS
Modify default settings for diagnosis object

The MODIFY-OBJECT-ASSUMPTIONS statement allows the user to modify the assumptions made automatically by DAMP regarding the data from the diagnosis object. Thus, for instance, it is possible to use a different Assembler instruction set from the one selected by DAMP for disassembly.

Format

```
MODIFY-OBJECT-ASSUMPTIONS

ADDRESSING-MODE = *UNCHANGED / *PARAMETERS(...)
  *PARAMETERS(...)
    | CONTROL-BLOCK = <name 1..32 with-under> / <structured-name 1..32>
    | ,MODE = *STD / *XS31 / *NXS
```

Operands

ADDRESSING-MODE = *UNCHANGED / *PARAMETERS(...)
The way in which the address fields are interpreted can be changed for the data structure specified under CONTROL-BLOCK. By default, DAMP interprets all addresses as 31-bit addresses (for /390 objects) or 32-bit addresses (for x86 objects). Switching to 24-bit or 31-bit addresses may be desirable in some cases.

ADDRESSING-MODE = *PARAMETERS(...)
The addressing mode for the data structure specified under CONTROL-BLOCK can be changed.

CONTROL-BLOCK = <name 1..32 with-under> / <structured-name 1..32>
This specifies the name of the data structure (DSECT, model etc.) to which the new addressing mode is to apply.

MODE = *STD / *XS31 / *NXS
This specifies the new addressing mode. The default value (*STD) specifies that DAMP is to interpret addresses as 31-bit addresses (for /390 objects) or 32-bit addresses (for x86 objects). *XS31 stands for 31-bit addressing and *NXS stands for 24-bit addressing.

Example

```
MODIFY-OBJECT-ASSUMPTION ADDRESSING-MODE=(ID1FCB,*NXS)
```

MODIFY-SCREEN-LAYOUT
Define new sequence and size for diagnostic windows

The MODFIY-SCREEN-LAYOUT statement is used to change the sequence and size of the diagnostic windows displayed on the screen.

Format

```
MODIFY-SCREEN-LAYOUT

 FIRST-WINDOW = *UNCHANGED / <integer 0..99>(...)
     <integer>(...)
         | SIZE = *UNCHANGED / <integer 2..19>
,SECOND-WINDOW = *UNCHANGED / <integer 0..99>(...)
     <integer>(...)
         | SIZE = *UNCHANGED / <integer 2..19>
,THIRD-WINDOW = *UNCHANGED / <integer 0..99>(...)
     <integer>(...)
         | SIZE = *UNCHANGED / <integer 2..19>
,FOURTH-WINDOW = *UNCHANGED / <integer 0..99>(...)
     <integer>(...)
         | SIZE = *UNCHANGED / <integer 2..19>
,FIFTH-WINDOW = *UNCHANGED / <integer 0..99>(...)
     <integer>(...)
         | SIZE = *UNCHANGED / <integer 2..19>
,SIXTH-WINDOW = *UNCHANGED / <integer 0..99>(...)
     <integer>(...)
         | SIZE = *UNCHANGED / <integer 2..19>
,SEVENTH-WINDOW = *UNCHANGED / <integer 0..99>(...)
     <integer>(...)
         | SIZE = *UNCHANGED / <integer 2..19>
```

(part 1 of 2)

```
,EIGHTH-WINDOW = *UNCHANGED / <integer 0..99>(...)
    <integer>(...)
      | SIZE = *UNCHANGED / <integer 2..19>
,NINTH-WINDOW = *UNCHANGED / <integer 0..99>(...)
    <integer>(...)
      | SIZE = *UNCHANGED / <integer 2..19>
```

(part 2 of 2)

Operands

FIRST-WINDOW = *UNCHANGED / <integer 0..99>(...)
...
NINTH-WINDOW = *UNCHANGED / <integer 0..99>(...)
The specified window is located at the position indicated by the operand name (FIRST-, SECOND-, ..., NINTH-WINDOW). For instance, specifying SECOND-WINDOW=5 causes window number 5 to be placed in the second position on the screen, provided that there is sufficient space on the screen. Windows 0 - 9 and 21 - 99 are supported; windows 10 - 20 are 'reserved'.

SIZE = *UNCHANGED / <integer 2..19>
A new size is defined for the specified window. It is possible to specify anything from 2 to 19 lines.

Example

```
MODIFY-SCREEN-LAYOUT FIRST-WINDOW = 9(SIZE=4),
                     SECOND-WINDOW = 33,
                     THIRD-WINDOW = 4
```

Restriction

If the operand names FIRST-WINDOW, ... NINTH-WINDOW are used in the statement, they must be used in an unbroken sequence and FIRST-WINDOW must be the first operand used.

This means that the following statements are not permitted:

```
MODIFY-SCREEN-LAYOUT THIRD-WINDOW = 1(SIZE=4)
MODIFY-SCREEN-LAYOUT FIRST-WINDOW = 4,
                     THIRD-WINDOW = 5
```

It is recommended that you use positional operands for the entries:

```
MODIFY-SCREEN-LAYOUT 9, 4, 6, 33(2)
```

OPEN-DIAGNOSIS-OBJECT
Open diagnosis object for processing

The OPEN-DIAGNOSIS-OBJECT statement assigns a diagnosis object (BS2000 dump, SELF-LOADER dump, PAM file, active BS2000 system) for diagnosis.

DAMP requires the standard BS2000 symbols for the relevant version in order to open a BS2000 object.

Format

```
OPEN-DIAGNOSIS-OBJECT

 OBJECT = *SYSTEM(...) / *#0(...) / *#1(...) /... / *#9(...) / <filename 1..54 without-gen-vers with-wild(80)>(...) /

    *SYSTEM(...)
      │ KIND-OF-SYSTEM = *BS2000

    *#0(...) / *#1(...) /... / *#9(...)
      │ KIND-OF-OBJECT = *STD / *BS2000 / *SELF-LOADER / *PAM

    <filename 1..54 without-gen-vers with-wild(80)>(...)
      │ KIND-OF-OBJECT = *STD / *BS2000 / *SELF-LOADER / *PAM

,SYMBOLS = *STD(...) / <filename 1..54 without-gen-vers>(...)

    *STD(....) / <filename 1..54>(...)
      │ ELEMENT = *BS2000(...) / <filename 1..54 without-cat-user-gen-vers>(...)
      │     *BS2000(...)
      │       │ VERSION= *STD / <composed-name_1..24_with-under>
      │     <filename 1..54 without-cat-user-gen-vers>(...)
      │       │ VERSION= *STD / <composed-name_1..24_with-under>
```

Operands

OBJECT = *SYSTEM(...) / *#0(...) / *#1(...) / ... / *#9(...) /
<filename 1..54 without-gen-vers with-wild(80)>(...)
This specifies the type of the diagnosis object.

OBJECT = *SYSTEM(...)
*SYSTEM assigns the active system as the diagnosis object. The user must have test authorization in order to access the active system. The system administrator can grant this authorization using the MODIFY-USER-ATTRIBUTES command and activate it by issuing the command MODIFY-TEST-OPTIONS PRIVILEG=*PAR(READ=8,WRITE=1) before the program is started.

KIND-OF-SYSTEM = *BS2000
This operand permits diagnosis in the current BS2000 system.

OBJECT = *#0(...) / *#1(...) / ... / *#9(...) /
<filename 1..54 without-gen-vers with-wild(80)>(...)
A file is assigned as a diagnosis object. The keywords *#0 through *#9 (or #0 through #9 for short) are interpreted as link names if the ADD-FILE-LINK command was issued beforehand with these link names.
In addition, all the wildcards permitted for SHOW-FILE-ATTRIBUTES may be specified. If a file can be uniquely identified using the wildcards specified or the partial qualification, it is opened. Otherwise, a message is issued stating that the specification must be qualified in more detail. If the file name contains the expression $TSN, it is replaced by the TSN of the task under which DAMP is currently executing.

KIND-OF-OBJECT = *STD / *BS2000 / *SELF-LOADER / *PAM
This passes the data structure of the diagnosis object to DAMP.

KIND-OF-OBJECT = *STD
For dump files with multiple objects (e.g. for a complete VM2000 SLED), the status window (W2) is initially displayed in the dialog in INF mode with the possible selection. The required diagnosis object can be selected by marking. Otherwise *STD has the same effect as *BS2000.

KIND-OF-OBJECT = *BS2000
DAMP searches in the dump file for a BS2000 object for diagnosis. If no BS2000 object is found, the dump file is opened as a self-loader.

KIND-OF-OBJECT = *SELF-LOADER
Any number of files with the structure of a dump file can be opened. DAMP does not provide any automatic functions for SELF-LOADER dumps. All addresses are interpreted as real addresses, so only real memory segments can be displayed in the normal formats (see also page 141). More detailed analyses can only be performed via PRODAMP procedures.

KIND-OF-OBJECT = *PAM
If a file does not have the same structure as a dump file, it can only be opened as a PAM file. The data in a PAM file can only be accessed using the methods described in the section “Processing files in PAM format” on page 140.

SYMBOLS = *STD(...) / <filename 1..54 without-gen-vers>(...)
The SYMBOLS operand must not normally be assigned a value.
This means that the default settings (SYMBOLS=*STD(*BS2000(*STD))) then take effect, which means that DAMP loads the BS2000 symbols that match the open BS2000 object from the standard library. If KIND-OF-OBJECT=*SELF-LOADER or *PAM was used, no symbols are loaded. If the SYMBOLS operand is used with other values, DAMP always tries to load symbols (also for KIND-OF-OBJECT=*SELF-LOADER or *PAM).

The SYMBOLS operand assigns the symbols that are required by DAMP to open the diagnosis object symbolically. Additional symbols for subsequent processing can be assigned with the ADD-SYMBOLS statement.

SYMBOLS = *STD(...) / <filename>(...)
The standard symbol library is selected as the library if *STD is specified.

If some other symbols are to be used for a BS2000 object or if special symbols are needed for a non-BS2000 object, a symbol library containing the required symbol element can be defined.

ELEMENT = *BS2000(...)
The BS2000 element name is selected. In the standard symbol library, this element name contains the symbols for the analysis of the BS2000 system.

VERSION = *STD
The element version that matches the open diagnosis object is selected.

VERSION = <composed-name_1..24_with-under>
Selects the specified element version.
The symbols in the standard symbol library have the same version name as the product for which they are valid.

ELEMENT = <filename 1..54 without-cat-user-gen-vers>(...)
This specifies the name of the element containing the symbols. The symbol elements supplied as standard with DAMP have the same name as the product to which they belong.

VERSION = *STD
The element version that matches the open diagnosis object is selected.

VERSION = <composed-name_1..24_with-under>
Selects the specified element version.
The symbols in the standard symbol library have the same version name as the product for which they are valid.

Examples

```
OPEN-DIAGNOSIS-OBJECT OBJECT=*SYSTEM

OPEN OBJECT=$SLED.SLED.1234,SYMBOLS=syssmb.damp.<ver>(BS2000(200))
```

PRINT-LIST
Start list output

The PRINT-LIST statement is used to output a set of objects previously selected by means of the statements START-LIST-GENERATION, ADD-LIST-OBJECTS or REMOVE-LIST-OBJECTS. Output can be directed to a file or to SYSLST.

Format

PRINT-LIST
OUTPUT = *SYSLST / <filename 1..54>

Operands

OUTPUT = *SYSLST / <filename 1..54>
This specifies the destination for the output.

OUTPUT = *SYSLST
By default, output is directed to SYSLST.

OUTPUT = <filename 1..54>
Output is directed to a cataloged file (SAM file).

Note

The following steps should be followed to print a list in batch mode:

```
START-LIST-GENERATION  ──────────────────────────────────────────  (1)
ADD-LIST-OBJECTS ...   ──────────────────────────────────────────  (2)
ADD-LIST-OBJECTS ...
REMOVE-LIST-OBJECTS ...
PRINT-LIST ...  ─────────────────────────────────────────────────  (3)
```

(1) Start list generation

(2) Define scope of list

(3) Print list

In interactive mode, the START-LIST-GENERATION statement displays the LIST window where the user can define the scope of the list by marking objects and filling in masks. In interactive mode, a corresponding field can be marked instead of issuing the PRINT-LIST statement.

PRINT-LOGGING-FILE
Start list output

The PRINT-LOGGING-FILE statement defines the output layout of a logging file and initiates printing of the file.

Format

```
PRINT-LOGGING-FILE

 LOGGING-FILE = <filename 1..54 without-gen-vers>
,OUTPUT-FILE = *STD /  <filename 1..54 without-gen-vers>
,LAYOUT = *STD / *TOMDOC / *OPTIONS(...)
    *OPTIONS(...)
      |  LINES-PER-PAGE = <integer 1..100>
      |,LOWER-CHARACTERS = *YES / *NO
      |,NIL-CHARACTERS = <x-string 1..2> / <c-string 1..1>
      |,TRASH-CHARACTERS = <x-string 1..2> / <c-string 1..1>
,PRINT = *NO / *YES / *ERASE
```

Operands

LOGGING-FILE = <filename 1..54 without-gen-vers>
Specifies the name of the logging file to be processed.

OUTPUT-FILE = *STD / <filename 1..54 without-gen-vers>
Specifies the name of the output file to be generated.
*STD means output to SYSLST.

LAYOUT = *STD / *TOMDOC / *OPTIONS(...)
Specifies the format of the output file.

LAYOUT = *STD
Sets default values for the format of the output file.

LAYOUT = *TOMDOC
The output file is to be edited using the control characters from the text editing system TOM-DOC.

LAYOUT = *OPTIONS(...)
The layout of the output file can be defined using the following options:

LINES-PER-PAGE = <integer 1..100>
Determines the number of lines to be printed per page.

LOWER-CHARACTERS = *YES / *NO
Specifies whether lowercase letters are to be printed (*YES) or whether they are to be converted to uppercase before printing (*NO).

NIL-CHARACTERS = <x-string 1..2> / <c-string 1..1>
Defines the representation of the NIL character.

TRASH-CHARACTERS = <x-string 1..2> / <c-string 1..1>
Defines the representation of the smudge character.

PRINT = *NO / *YES / *ERASE
Specifies whether the generated output file is to be printed and whether it should subsequently be deleted.

PRINT = *NO
The generated output file is not printed. This is the default value.

PRINT = *YES
The generated output file is printed but not subsequently deleted.

PRINT = *ERASE
The generated output file is printed and is subsequently deleted.

Examples

```
PRINT-LOGGING-FILE LOGGING-FILE=LOG.HUGO,OUTPUT-FILE=LOG.ABC.EDITED,
                   PRINT=*ERASE
```

REMOVE-LIST-OBJECTS
Control list output

The REMOVE-LIST-OBJECTS statement allows users to exclude from output areas previously assigned by means of the ADD-LIST-OBJECTS command.
The options available for canceling selected objects are, however, of a relatively general nature. If only specific areas are to be excluded from output for a given task, the entire task must first be canceled using the REMOVE-LIST-OBJECTS statement. The areas to be output must then be assigned by way of the ADD-LIST-OBJECTS statement. The areas not required for output are not assigned during this process.

Format

```
REMOVE-LIST-OBJECTS

 GLOBAL-INFORMATION = *NONE / *ALL / *TRACES / *MAPS / *CONTROL-BLOCKS /
                      *MEMORY-AREAS / *PAGES / *MODULE
,TASK-INFORMATION = *NONE / *ALL / <x-string 1..8> / <alphanum-name 1..4> / <c-string 1..4>
,USER-LIST-PROCEDURE = *NONE / *ALL / <name 1..32 with-under> / <structured-name 1..32>
,WINDOW = *NONE / *ALL / <integer 4..99>
,DUMP-DIAGNOSIS = *NONE / *ALL
```

Operands

GLOBAL-INFORMATION = *NONE / *ALL / *TRACES / *MAPS / *CONTROL-BLOCKS / *MEMORY-AREAS / *PAGES / *MODULE
Output of global information can be suppressed entirely or for specific areas.

GLOBAL-INFORMATION = *NONE
If no selection is made, the areas declared by way of ADD-LIST-OBJECTS are output.

GLOBAL-INFORMATION = *ALL
This specification excludes not only the areas declared by means of ADD-LIST-OBJECTS, but also the default areas (see brief report).

GLOBAL-INFORMATION = *TRACES
The global traces declared by means of ADD-LIST-OBJECTS are not output.

GLOBAL-INFORMATION = *MAPS
Information on the loaded CSECTs and the correction status of the system selected by the ADD-LIST-OBJECTS statement is not output.

GLOBAL-INFORMATION = *CONTROL-BLOCKS
All control blocks selected for output by the ADD-LIST-OBJECTS statement are not output.

GLOBAL-INFORMATION = *MEMORY-AREAS
The areas of the virtual address space selected for output by the ADD-LIST-OBJECTS statement are not output.

GLOBAL-INFORMATION = *PAGES
Pages or areas of the address space (virtual, main memory, HSA, ...) selected for output by the ADD-LIST-OBJECTS statement are not output.

GLOBAL-INFORMATION = *MODULE
The memory area of the specified module selected for output by the ADD-LIST-OBJECTS statement is not output.

TASK-INFORMATION = *NONE / *ALL / <x-string 1..8 > /<alphanum-name 1..4> / <c-string 1..4>
Output of task-specific information can be suppressed entirely or for individual tasks.

TASK-INFORMATION = *NONE
If no selection is made, the information declared in ADD-LIST-OBJECTS for the specified tasks is output.

TASK-INFORMATION = *ALL
This specification excludes not only the information declared in ADD-LIST-OBJECTS for the tasks specified there, but also the information for the default tasks.

TASK-INFORMATION = <x-string 1..8 >
The specified hexadecimal value is interpreted as the TID of the task for which information is not to be output.

TASK-INFORMATION = <alphanum-name 1..4>
The specified alphanumeric name is interpreted as the TSN of the task for which information is not to be output.

TASK-INFORMATION = <c-string 1..4 >
The specified character string is interpreted as the TSN of the task for which information is not to be output.

USER-LIST-PROCEDURE = *NONE / *ALL / <name 1..32 with-under> <structured-name 1..32>
This operand specifies the name of a PRODAMP program located in the current PRODAMP object library. This program is to be started automatically when list output is complete.

USER-LIST-PROCEDURE = *NONE
If no specification is made, the user programs declared by means of ADD-LIST-OBJECTS are started automatically after the list has been output.

USER-LIST-PROCEDURE = *ALL
None of the user programs declared with ADD-LIST-OBJECTS are started when list output is complete.

USER-LIST-PROCEDURE = <name 1..32 with-under> / <structured-name 1..32>
The specified user program is not started when list output is completed.

WINDOW = *NONE / *ALL / <integer 4..99>
This specifies a window whose layout was selected for output to a list using the ADD-LIST-OBJECTS statement.
The user can exclude all windows or specify individual windows for exclusion.

WINDOW = *NONE
All the windows declared by means of ADD-LIST-OBJECTS are output.

WINDOW = *ALL
None of the windows declared by means of ADD-LIST-OBJECTS is output.

WINDOW = <integer 4..99>
The specified window is not output. Windows 4 - 9 and 21 - 99 are supported; windows 10 - 20 are 'reserved'.

DUMP-DIAGNOSIS = *NONE / *ALL
Invocation of automatic preanalysis can be deactivated (*ALL).
The information set in the status window (W2) in INF mode can be output.

Example

```
REMOVE-LIST-OBJECTS TASK-INFORMATION=X'0004000E',WINDOW=*ALL
```

Once the PRINT-LIST statement has been issued, the selection of the areas to be output is reset to the default. However, the scope of the standard output list can be restricted even further by means of the REMOVE-LIST-OBJECTS statement (see the description of the GLOBAL-INFORMATION and TASK-INFORMATION operands).

REPEAT-SESSION
Replay diagnostics log

The REPEAT-SESSION statement is used to replay the inputs and outputs made during a DAMP session and recorded in a file.

Format

REPEAT-SESSION
LOGGING-FILE = <filename 1..54>

Operands

LOGGING-FILE = <filename 1..54>
Specifies the name of the file containing the log to be replayed.

Example

```
REPEAT-SESSION LOGGING-FILE=LOG.HUGO
```

RESUME-PRODAMP-PROGRAM
Resume interrupted PRODAMP program

The RESUME-PRODAMP-PROGRAM statement resumes an interrupted PRODAMP program. If the program was not interrupted, it is started from from the beginning.

Format

RESUME-PRODAMP-PROGRAM
NAME = *INTERRUPTED / <integer 4..99> / <structured-name 1..32 / <name 1..32 with-under> ,PARAMETERS = *NONE / list-poss(32): <integer -2147483648..2147483647> / <x-string 1..8> / <c-string 1..80>

Operands

NAME = *INTERRUPTED / <integer 4..99> / <structured-name 1..32> / <name 1..32 with-under>
Specifies the PRODAMP program to be resumed.

NAME = *INTERRUPTED
If a number of programs have been interrupted, the program most recently interrupted is resumed.

NAME = <integer 4..99>
Specifies the number of the window to be assigned to the PRODAMP compiler. The object already generated by the compiler is to be resumed. Windows 4 - 9 and 21 - 99 are supported; windows 10 - 20 are “reserved”.

NAME = <structured-name 1..32> / <name 1..32 with-under>
Specifies the name of a program already started with this name using the START-PRODAMP-PROGRAM statement or which has been compiled without errors under this name by the PRODAMP compiler.

PARAMETERS = *NONE / list-poss(32): <integer -2147483648..2147483647> / <x-string 1..8> / <c-string 1..80>
This operand allows a list of up to 32 parameters to be passed to the interrupted PRODAMP program. The parameters can be declared in numeric or hexadecimal format or as a character string. See page 258 for details on how the PRODAMP program takes over the values passed.

Examples

```
RESUME-PRODAMP-PROGRAM NAME=*INTERRUPTED, PAR=(X'10000','HUGO')
RESUME-PRODAMP-PROGRAM PROC1,(12,13,14)
```

SEARCH-IN-SUBSYSTEM
Perform CSECT search in subsystem

Function

The SEARCH-IN-SUBSYSTEM statement restricts the CSECT search to a single subsystem. This restriction can be undone with the same statement. If no subsystem version is specified, the first subsystem from the subsystem list is used. The unique context name can be specified as an alternative to specifying SUBSYSTEM/VERSION.

All subsystems are still taken into account for the qualification of addresses.

Format

```
SEARCH-IN-SUBSYSTEM

SUBSYSTEM = *ALL / <name 1..8>(...) /<c-string 1..8>(...) / *CONTEXT(...)
    <name 1..8>(...)
        VERSION = *FIRST-FOUND / <filename 1..8> / <c-string 1..8>
    *CONTEXT(...)
        CONTEXT = <text 1..32>
```

Operands

SUBSYSTEM = *ALL / <name 1..8>(...) / <c-string 1..8>(...) / *CONTEXT(...)
Specifies the name of the subsystem..

SUBSYSTEM = *ALL
This entry undoes a previously imposed restriction.

SUBSYSTEM = <name 1..8>(...) / <c-string 1..8>(...)
Specifies the name of the subsystem as displayed in the SUSY window.

VERSION = *FIRST-FOUND / <filename 1..8> / <c-string 1..8>
Specifies the version of the selected subsystem. If no subsystem version is specified, the first subsystem in the subsystem list with the specified name is used.

SUBSYSTEM = *CONTEXT(...)
Specifies the context name corresponding to the subsystem.

CONTEXT = <text 1..32>
The context name specified here can be taken from the SUSY window (see the CTX setting on page 115).

SHOW-EDITED-INFORMATION
Output edited diagnostic data

The SHOW-EDITED-INFORMATION statement is used to output edited diagnostic data to the specified dump window.

Format

```
SHOW-EDITED-INFORMATION

 INFORMATION = *STORAGE-EDIT / *AUDIT-TABLE-EDIT / *TRACE-TABLE-EDIT / *TASK-TABLES /
                       *SUBSYSTEM-INFORMATION / *DUMPED-SYSTEM-FILE / *MEMORY-ATTRIBUTES /
                       *SPECIAL(...)

    *SPECIAL(...)
       |  NAME = <structured-name 1..255>

,WINDOW = *NEXT-FREE / <integer 4..99>
```

Operands

INFORMATION = *STORAGE-EDIT / *AUDIT-TABLE-EDIT / *TRACE-TABLE-EDIT / *TASK-TABLES / *SUBSYSTEM-INFORMATION / *DUMPED-SYSTEM-FILE / *MEMORY-ATTRIBUTES / *SPECIAL(...)
Specifies the method used to edit the data.

INFORMATION = *STORAGE-EDIT
The data is edited using the layout of the standard DAMP window. Any special editing method previously selected for the data will be canceled.

INFORMATION = *AUDIT-TABLE-EDIT
If AUDIT tables (hardware AUDIT or linkage AUDIT) are contained in a dump, the first table found is edited and displayed in the specified window.
The AUDIT window allows you to display other AUDIT tables subsequently. See page 123 for further details.

INFORMATION = *TRACE-TABLE-EDIT
In the case of SLEDs and SNAP dumps, the entire trace table is edited, while in the case of system dumps the task-specific trace table is edited. The table is then displayed in the specified window. See page 108 for further details.

INFORMATION = *TASK-TABLES
The contents of a number of freely-selectable fields are displayed in the form of a tabular overview for all the tasks active in the system. See page 111 for further details.

INFORMATION = *SUBSYSTEM-INFORMATION
Selected data is edited and output for the subsystems currently active in the system. See page 113 for further details.

INFORMATION = *DUMPED-SYSTEM-FILE
This displays, edits and generates system files and dump sections. See page 120 for further details.

INFORMATION = *MEMORY-ATTRIBUTES
This provides information on memory allocation and memory attributes of the current address space. See page 111 for further details.

INFORMATION = *SPECIAL(...)
This operand provides the option of special ad hoc formats. You must specify the name of a dynamically loadable DAMP module which implements this function. The module is then dynamically loaded from the library SYSLNK.DAMP.<ver>.

NAME = <structured-name 1..255>
This specifies the name of the module in the library SYSLNK.DAMP.<ver> which implements the required special function.
This operand is only supported for reasons of compatibility.

WINDOW = *NEXT-FREE / <integer 4..99>
This specifies the window in which the edited diagnostic data is to be output.
Windows 4 - 9 and 21 - 99 are supported; windows 10 - 20 are 'reserved'.
If *NEXT-FREE is specified, the windows 4 to 9 are used in the following order: 9, 8, ..., 4.

Examples

```
SHOW-EDITED-INFORMATION INFORMATION=*TRACE-TABLE-EDIT, WINDOW=99
```

SHOW-LAST-STATEMENT
Display last DAMP statement

The statement SHOW-LAST-STATEMENT redisplays the most recently issued DAMP statement in the DAMP command line. This allows the statement to be modified if necessary and then issued again.

This statement has no operands.

Multiple input of the SHOW-LAST-STATEMENT statement enables you to scroll back in the statement history.

SHOW-PRODAMP-LIBRARIES
Display PRODAMP libraries

The SHOW-PRODAMP-LIBRARIES statement displays the names of the user libraries currently assigned for PRODAMP in the message lines of the DAMP screen.

This statement has no operands.

SHOW-SUBSYSTEM-FOR-SEARCH
Display currently set subsystem

Function

The SHOW-SUBSYSTEM-FOR-SEARCH statement displays the subsystem set by means of SEARCH-IN-SUBSYSTEM.

This statement has no operands.

START-LIST-GENERATION
Prepare list output

The START-LIST-GENERATION statement is used to prepare for list output. The areas to be output are selected in the LIST window or using the ADD-LIST-OBJECTS and REMOVE-LIST-OBJECTS statements. List output is actually started using the PRINT-LIST statement.

Format

```
START-LIST-GENERATION

 FILES-TO-EVALUATE = *CURRENT / *#0 / *#1 / ...  / *#9 /
                              <filename 1..54 without-gen-vers with-wild(80)>(...)
    <filename>(...)
       │ EVALUATE = *FIRST-MATCH / *ALL-MATCHES
,WINDOW = *NEXT-FREE / <integer 4..99>
```

Operands

FILES-TO-EVALUATE = *CURRENT / *#0 / *#1 / ... / *#9 /
<filename 1..54 without-gen-vers-with-wild(80)>(...)
This specifies the files on which list editing is to be performed.
This operand has no effect in interactive mode.

FILES-TO-EVALUATE = *CURRENT
The dump file which is currently open (*CURRENT) is used. This is the default value.

FILES-TO-EVALUATE = *#0 / *#1 / ... / *#9
The file that was assigned to the selected link name with the ADD-FILE-LINK command is used.

FILES-TO-EVALUATE = <filename 1..54 without-gen-vers-with-wild(80)>(...)
It is possible to select more than one file by specifying a fully qualified file name including wildcards.

EVALUATE = *FIRST-MATCH / *ALL-MATCHES
If more than one file matches the file name specified using wildcards, the EVALUATE operand specifies whether only the first file is to be evaluated (*FIRST-MATCH) or whether all matched files are to be evaluated (*ALL-MATCHES). The default setting is that only the first matching file is evaluated.

WINDOW = *NEXT-FREE / <integer 4..99>
This operand is only permitted in interactive mode
This specifies the window in which the selection mask for generating a list is to be displayed (the LIST window).
The default value (*NEXT-FREE) specifies that the next free window is to be used as the LIST window. Windows 4 - 9 and 21 - 99 are supported; windows 10 - 20 are 'reserved'.
If *NEXT-FREE is specified, the windows 4 to 9 are used in the following order: 9, 8, ..., 4.

Examples

```
START-LIST-GENERATION FILES=$SYSDUMP.*DCLOSE*(EVAL=*ALL)

S-L-G FILES=*CURRENT, WINDOW=98
```

START-MODULE
Start external subroutine

The START-MODULE statement calls an external subroutine using the VMOS linkage. If the subroutine is not yet loaded, DAMP attempts to load it from the SYSLNK.DAMP.<ver> library. The LOAD-MODULE statement must be used to inform DAMP of the location of subroutines from other libraries. If an error occurs in a subroutine, the subroutine is aborted, but not by DAMP.
If the subroutine is in an endless loop, it can be aborted by pressing K2 and then entering the BS2000 command INFORM-PROGRAM MSG='*CANCEL'. DAMP then continues normally. See also the LOAD-MODULE statement on page 191.

Format

START-MODULE
NAME = <name 1..8> ,PARAMETERS = <cmd-rest 0..4096>

Operands

NAME = <name 1..8>
This specifies the name of the external subroutine to be called.

PARAMETERS = <cmd-rest 0..4096>
This defines a character string which is passed unchanged to the subroutine, i.e. the program to be called is passed the address of the string in register R1.

Example

```
START-MODULE NAME=DCM, PARAMETERS=WHATEVER YOU WANT
```

START-OPTION-DIALOG
Set user options

The START-OPTION-DIALOG statement displays a selection mask (OPTIONS window) where the user can set the global default values for DAMP.
The START-OPTION-DIALOG statement is only permitted in interactive mode.

See page 135 for further details.

Format

START-OPTION-DIALOG
WINDOW = *NEXT-FREE / <integer 4..99>

Operands

WINDOW = *NEXT-FREE / <integer 4..99>
This specifies the window in which the selection mask for setting the default values is to be displayed (the OPTIONS window).
The default value (*NEXT-FREE) specifies that the next free window is to be used as the OPTIONS window. Windows 4 - 9 and 21 - 99 are supported; windows 10 - 20 are 'reserved'.
If *NEXT-FREE is specified, the windows 4 to 9 are used in the following order: 9, 8, ..., 4.

Example

```
START-OPTION-DIALOG WINDOW=4
```

START-PATTERN-SEARCH
Prepare string search

The START-PATTERN-SEARCH statement displays the selection mask for a selective string search in the required window.

See page 125 for further details.

Format

START-PATTERN-SEARCH
WINDOW = *NEXT-FREE / <integer 4..99>

Operands

WINDOW = *NEXT-FREE / <integer 4..99>
This specifies the window in which the selection mask (FIND window) is to be displayed. The default value (*NEXT-FREE) specifies that the next free window is to be used. Windows 4 - 9 and 21 - 99 are supported; windows 10 - 20 are 'reserved'.
If *NEXT-FREE is specified, the windows 4 to 9 are used in the following order: 9, 8, ..., 4.

Example

```
START-PATTERN-SEARCH WINDOW=5
```

START-PRODAMP-EDITOR
Load editor for PRODAMP compiler

The START-PRODAMP-EDITOR statement assigns a special DAMP window (PRODAMP window). This window can then be used to edit, compile and execute PRODAMP procedures.
It is possible to read a PRODAMP source into this window from a file when the window is called (see the SOURCE operand).
For details on working with the PRODAMP compiler, see section “Working with procedures (special window: PROC)” on page 299.

Format

START-PRODAMP-EDITOR
WINDOW = *NEXT-FREE / <integer 4..99> ,SOURCE = *NONE / <filename 1..54>

Operands

WINDOW = *NEXT-FREE / <integer 4..99>
This specifies the window to be used to display the PRODAMP window.
The default value (*NEXT-FREE) specifies that the next free window is to be used as the PRODAMP window. Windows 4 - 9 and 21 - 99 are supported; windows 10 - 20 are 'reserved'.
If *NEXT-FREE is specified, the windows 4 to 9 are used in the following order: 9, 8, ..., 4.

SOURCE = *NONE / <filename 1..54>
This specifies the name of a file whose contents act as the source to be read into the PRODAMP window. The default setting does not assign a file.

Example

```
START-PRODAMP-EDITOR WINDOW=6, SOURCE=MY.PRODAMP.SOURCE
```

START-PRODAMP-PROGRAM
Load and start PRODAMP program

The START-PRODAMP-PROGRAM statement loads and starts PRODAMP program from a PRODAMP library.

Format

```
START-PRODAMP-PROGRAM

 NAME = <name 1..32 with-under> / <structured-name 1..32> / *LIBRARY-ELEMENT(...)

    *LIBRARY-ELEMENT(...)
       | LIBRARY = *PRODAMP-USER-OBJECT-LIBRARY / *PRODAMP-SYSTEM-LIBRARY /
       |             <filename 1..54 without-gen-vers>
       | ELEMENT = <name 1..32 with-under> / <structured-name 1..32>

,PARAMETERS = *NONE / list-poss(32): / <integer -2147483648..2147483647> / <x-string 1..8> /
                <c-string 1..80>
```

Operands

NAME = <name 1..32 with-under> / <structured-name 1..32> / *LIBRARY-ELEMENT(...)
Designates the name of the PRODAMP program.

NAME = <name 1..32 with-under> / <structured-name 1..32>
Loads and starts a PRODAMP program from the currently set PRODAMP user object library. This name is identical to the name of the library element.

NAME = *LIBRARY-ELEMENT(...)
Loads and starts a PRODAMP program from the selected PRODAMP library. The name of the program is identical to the name of the library element.

LIBRARY = *PRODAMP-USER-OBJECT-LIBRARY / *PRODAMP-SYSTEM-LIBRARY / <filename 1..54 without-gen-vers>
Designates the PRODAMP library from which the program is to be loaded.

LIBRARY = *PRODAMP-USER-OBJECT-LIBRARY
Loads the PRODAMP program from the currently set PRODAMP user object library.

LIBRARY = *PRODAMP-SYSTEM-LIBRARY
Loads the PRODAMP program from the PRODAMP system library. For standard installations, this is the library $TSOS.SYSDMP.DAMP.

LIBRARY = <filename 1..54 without-gen-vers>
Loads the PRODAMP program from the specified PRODAMP library.

ELEMENT = <name 1..32 with-under> / <structured-name 1..32>
Designates an element from a PRODAMP library. The name of the element is identical to the name of the PRODAMP program that is to be loaded and started.

PARAMETERS = *NONE / list-poss(32): <integer -2147483648..2147483647> / <x-string 1..8> / <c-string 1..80>
This operand can be used to pass a list of up to 32 parameters to the PRODAMP program. The parameters can be defined numerically or as hexadecimal or character strings. For more details on transferring values to the PRODAMP program, see page 258.

Example

```
START-PRODAMP-PROGRAM NAME=TEST, PARAMETER = (1,2,X'ED','HUGO')
```

Notes

– In contrast to the ASSIGN-PRODAMP-LIBRARIES statement, the *LIBRARY-ELEMENT operand switches the PRODAMP user object library only for the duration of the PRODAMP program; on exiting the program, the earlier state is restored.

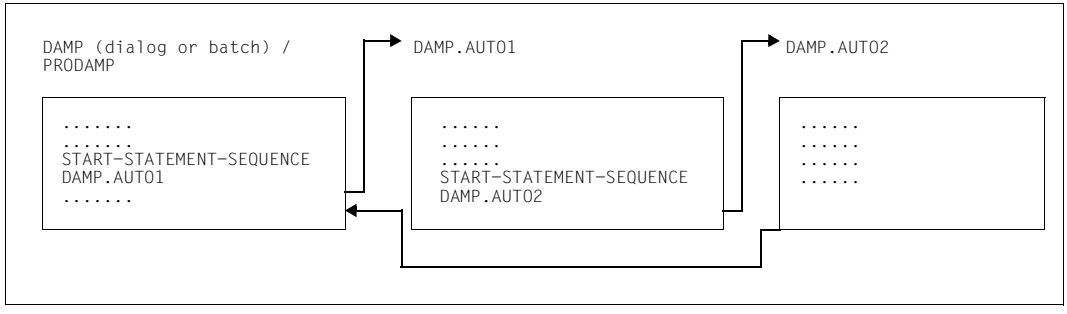
– On terminating the PRODAMP program started with START-PRODAMP-PROGRAM, all PRODAMP objects loaded for this run are unloaded. Note, however, that further objects may have been dynamically loaded due to subroutine calls. The objects are not unloaded on interrupting the program with the PRODAMP statement INTERRUPT.

– However, when a PRODAMP program is executed or compiled without errors in a PROC window (see page 300), this object and the dynamically loaded objects are not unloaded until the associated PROC window is closed or “New” is entered in the mode field.

– If, when loading a PRODAMP object, DAMP determines that an object of the same name is already loaded, the object is not reloaded. This also applies to the implicit dynamic loading of objects on calling a subroutine.

START-STATEMENT-SEQUENCE
Read DAMP statements from file

The START-STATEMENT-SEQUENCE statement reads DAMP statements from a file and then executes them. All output from DAMP is suppressed until the last statement has been read.
If the file itself in turn contains a START-STATEMENT-SEQUENCE statement calling a statement sequence from a different file, statements are then read from this second file. The remaining statements in the original file are, however, no longer executed. For this reason, it only makes sense to use the START-STATEMENT-SEQUENCE statement as the last statement in a file of this sort.



Format

START-STATEMENT-SEQUENCE
FILENAME = <filename 1..54>

Operands

FILENAME = <filename 1..54>
Specifies the file from which the DAMP statements are to be read.

STOP-LOGGING
Terminate logging of diagnosis run

The STOP-LOGGING statement terminates the logging of the diagnosis run started with the LOG-SESSION statement.

This statement has no operands.

USE-REGISTER
Define register use for disassembled output

The USE-REGISTER statement specifies that a particular register in the specified module is to be taken as the base for an area of the module or for a particular control block (DSECT) in the event of disassembled output. This means that the instructions are no longer specified using the base/offset format, but as an relative offset within the module or via the field names from the relevant control block.

The USE-REGISTER statement and the corresponding DROP-REGISTER statement do not support the output of disassembled x86 code. x86 registers may not be specified.

Format

```
USE-REGISTER

 MODULE-NAME = <name 1..32>
,REGISTER = <integer 0..15>
,FOR = *MODULE-BASE(...) / *CONTROL-BLOCK(...)
    *MODULE-BASE(...)
      | DISPLACEMENT = 0 /<integer -2147483648..2147483647> / <x-string 1..8>
    *CONTROL-BLOCK(...)
      | NAME = <structured-name 1..32> / <name 1..32 with-under>
```

Operands

MODULE-NAME = <name 1..32>
Specifies the name of the module to which the declaration for the disassembled output is to apply.

REGISTER = <integer 0..15>
Specifies the register to be used as the base register.

REGISTER = <integer 0..15>
Specifies a /390 general register.

FOR = *MODULE-BASE(...)
The register is to be used as the base register for the specified module.

DISPLACEMENT = 0 / <integer -2147483648..2147483647> / <x-string 1..8>
Specifies the relative starting address of the area (within the specified module) for which the register is to be used as the base.

FOR = *CONTROL-BLOCK(...)
The register is to be used as the base register for a control block.

NAME = <structured-name 1..32> / <name 1..32 with-under>
Specifies the name of the control block for which the register is to be used as the base.

Examples

```
USE-REGISTER MODULE-NAME=DOPEN, REGISTER=10,
FOR=*MODULE-BASE(DISPLACEMENT=X'1000')

USE-REGISTER MODULE=DCLOSE, REG=4, FOR=*CONTROL-BLOCK(NAME=EXVT)
```

5.6.2 System level

DAMP statements via the system command INFORM-PROGRAM

If DAMP is interrupted using the K2 key, DAMP functions can also be addressed via the BS2000 command INFORM-PROGRAM.
The MSG operand of the command is used to communicate with DAMP.

Format: INFORM-PROGRAM

```
INFORM-PROGRAM

 MSG = *NO / <c-string 1..64>
,JOB-IDENTIFICATION = *OWN / *TSN(...) / *MONJV(...)
    *TSN(...)
      | TSN = <alphanum-name 1..4>
    *MONJV(...)
      | MONJV = <filename 1..54 without-gen>
```

Input in system mode is as follows: `/INFORM-PROGRAM MSG='<function>'`

Format: MSG = '<function>'

```
INFORM-PROGRAM MSG='<function>'

 FUNCTION = *RESUME / HALT / *END / *TERMINATE / *DUMP / *TERMD / *ESCAPE / *BREAK /
              *CANCEL / *LOG-SESSION / *STOP-LOGGING / *REPEAT-SESSION(...) /
              *MODIFY-PAGE-ACCESS(...)
    *REPEAT-SESSION(...)
      | LOGGING-FILE = <filename 1..54>
    *MODIFY-PAGE-ACCESS(...)
      | AREA = <x-string 1..8> / <name 1..8>(...)
      |    <name>(...)
      |      | DISPLACEMENT = 0 / <integer 0..9> / <x-string 1..8>
      | ,STATE = *READ-ONLY / *WRITEABLE
```

Operands

FUNCTION = *RESUME / *HALT / *END / *TERMINATE / *DUMP / *TERMD / *ESCAPE / *BREAK / *CANCEL / *LOG-SESSION / *STOP-LOGGING / *REPEAT-SESSION(...) / *MODIFY-PAGE-ACCESS(...)
Indicates the functions which can be assessed using the INFORM-PROGRAM command.

FUNCTION = *RESUME
DAMP is resumed at the point it was interrupted with the K2 key.

FUNCTION = *HALT / *END / *TERMINATE
This terminates DAMP without a memory dump.
If DAMP is terminated abnormally and if a START-MODULE statement was active at the time of the interruption, the command /INFORM-PROGRAM MSG='*HALT' only aborts the external subroutine. DAMP can be resumed.

FUNCTION = *DUMP / *TERMD
This terminates DAMP with a memory dump.

FUNCTION = *ESCAPE / *BREAK
This displays the point at which DAMP was interrupted by K2.

FUNCTION = *CANCEL
This aborts any current DAMP function.

FUNCTION = *LOG-SESSION
This activates logging.

FUNCTION = *REPEAT-SESSION(...)
This replays the diagnostics log.

LOGGING-FILE = <filename 1..54>
This specifies the name of the file in which the inputs and outputs from the DAMP session were logged.

FUNCTION = *STOP-LOGGING
This deactivates logging.

FUNCTION = *MODIFY-PAGE-ACCESS(...)
This changes the status of memory pages in the address space in which DAMP is running.

AREA = <x-string 1..8> / <name 1..8>(...)
This specifies an area in the virtual address space for which the attributes are to be changed.

AREA = <x-string 1..8>
A hexadecimal string is interpreted as the number of a 4-Kb page.

AREA = <name 1..8>(...)
An alphanumeric entry is interpreted as a module name from the DAMP system.

DISPLACEMENT = 0 / <integer 0..9> / <x-string 1..8>
This specifies an address within the module relative to the start of the module. It is only possible to change the attributes of a page where the specified address lies between the beginning and end of the page.

STATE = *READ-ONLY / *WRITEABLE
This specifies whether or not it is to be possible to overwrite the specified memory area.

INFORM-PROGRAM MSG='?' activates DAMP's SDF user guidance mode; DAMP treats "FUNCTION" like any other DAMP statement.

Examples

```
/INFORM-PROGRAM MSG='FUNCTION=*CANCEL'
```

Entering system commands in the command line

Any BS2000 system command can be entered in the command line. The user alone is responsible for ensuring that no commands are issued which would terminate the program (e.g. EXIT-JOB).
If abbreviated commands are entered and the command abbreviation corresponds to an abbreviated DAMP statement, the DAMP statement is always executed. You can have the input interpreted as a system command by prefixing a label, for example, /.LABEL.
System command outputs are displayed in the DAMP message lines.

5.7 PRODAMP

5.7.1 Introduction

PRODAMP (PROcedure language for DAMP) is a language similar to Pascal for the formulation of diagnostic algorithms in DAMP. PRODAMP runs under DAMP and utilizes the functions offered there, such as symbolic addressing of data structures or output in various formats in screen windows.

With PRODAMP, it is possible to write decision-based statements, which would otherwise have to be entered individually by hand, into a procedure and to execute them automatically. It is possible, for example, to follow chains down to a structure which contains a specific data item, to search tables and process (e.g. arithmetically) the values they contain, or to have questions such as “Is this task holding a lock?” answered automatically.

Example

Let us assume that you frequently have to analyze problems where the program crashes due to a DMS error, and where the only aid at your disposal is a user dump. In order to identify which file and which DMS error code are involved, the following basic steps are necessary:

1. Assign a dump file.
2. Select the PCB which issued the DMS macro.
3. Mark register 1 in this PCB.
4. Assign the area (FCB) addressed by register 1 to another window.
5. Overlay this area with the DSECT of the FCB.
6. Position to the field ID1FILE (file name).
7. Position to the field ID1ECB (error code).

Steps 3 through 7 can be written as a PRODAMP procedure and subsequently executed automatically whenever it is needed.

For example, if the procedure was stored under the name “DMSERR”, the overhead involved in determining the DMS error code is reduced to the following steps:

1. Assign the dump file.
2. Select the PCB which issued the DMS macro (to provide values for CURRENT.PCB).
3. Issue the DAMP statement START-PRODAMP-PROGRAM DMSERR.

The PRODAMP procedure DMSERR would look something like this:

```
FNAM := ' '*54;
ERR := 0;
P := CURRENT.PCB;
FCB@ := P.ESTKGR1;
FNAM := FCB@.ID1FILE;
ERR := FCB@.ID1ECB ;
MESSAGE ( 'DMS ERROR '+HEX_STRING(ERR)+' FOR FILE '+FNAM );
```

This procedure offers even more than the individual steps listed above: the desired information is edited and displayed on the screen, saving you the bother of searching through the contents of the output window.

A procedure written in the PRODAMP language (PRODAMP source code) cannot be executed directly, but must first be compiled to form a PRODAMP program (PRODAMP object code). Compilation can be carried out in a PRODAMP window, which also provides a complete development environment for PRODAMP procedures. This window is called using the DAMP statement START-PRODAMP-EDITOR. In this window, you can store source code and generated objects in a PRODAMP library as well as perform editing and compilation operations and run procedures on an ad hoc basis. The DAMP statement START-PRODAMP-PROGRAM also allows you to run PRODAMP programs from a PRODAMP
library.

PRODAMP can thus handle routine tasks which have to be executed before you reach the actual heart of the problem.

5.7.2 Syntax

The PRODAMP syntax is very similar to that of standard block-oriented languages such as Pascal, Modula-2 or Ada, but has the advantage of being simpler. There is, for example, no declaration division for types, variables and constants; this makes it possible to create ad hoc algorithms for solving special problems without having to formulate them as precisely as is necessary for normal programs.

The syntax reflects the fact that, in addition to the familiar language elements found in the above-mentioned programming languages, PRODAMP contains language elements which are tailor-made for special diagnostic requirements. PRODAMP can, for example, address the diagnostic data symbolically, using field names taken from the associated symbol file. You can also modify the meta-characteristics of such symbols or define the formats of the output windows.

A complete definition of the can be found in section “Syntax diagrams” on page 312.

5.7.3 Language elements

5.7.3.1 Lexical elements

Character set

The character set of PRODAMP consists of

– letters
– special characters
– digits
– separators.

With regard to letters, string literals and comments are the only case where PRODAMP makes a distinction between uppercase and lowercase.

Separators
Whenever the names, numbers, etc. in a sequence are not separated by special characters, such characters must be inserted. Possible separator characters are blanks and comments.

A comment can contain any characters and is delimited by double quotes ("). Comments do not affect program execution; they serve simply to facilitate reading.

Names

Names (identifiers) are required in order to identify the various objects which can be used in a procedure (variables, subroutines, etc.).

They are made up of letters, digits, the characters $, # or @ and the underscore character ("_"), which can only be present once.

The first character must be a letter or one of the characters $, # or @, and the last character must not be an underscore. Furthermore, names must not be word symbols (operator, name of a statement, etc.). Names can be up to 31 characters in length.

Examples

```
HUGO
X123
A_ONE
@LABEL
T#1234
THIS_IS_AN_EXTREMELY_LONG_NAME
```

Length of source code lines

End-of-line characters have no meaning in the PRODAMP language. Nevertheless, the following rules should be observed:

Program lines should only be created with a maximum of 72 characters in the editor. If a line contains less than 72 lines, it must not end in the middle of an identifier, a literal etc.

When the PRODAMP source code is loaded into a PRODAMP window for compilation, line breaks are added to lines with more than 72 characters. Then lines with less than 72 characters are padded with blanks until they are 72 characters in length and lowercase characters outside strings and comments are converted to uppercase.

5.7.3.2 Operators

The following operators are permitted in PRODAMP:

= (equal to)
<> (not equal to)
< (less than)
<= (less than or equal to)
> (greater than)
>= (greater than or equal to)
+ (plus)
- (minus)
***** (times)
/ (divided by) and
MOD (modulo operation)

Two or more conditions can also be combined using the logical operators:

AND,
OR and
NOT

Since the operators have different priorities (see page 231), it may be necessary to use parentheses to achieve the desired effect. The operators AND and OR operate in short-circuit mode, i.e. evaluation of the condition is terminated as soon as its truth value has been determined.

The following operator is available for bit pattern expressions:

IN

This operator can be used to test single bits and bit combinations. It returns the value TRUE if and only if all the bits tested are set in the bit pattern.

Priority of the operators

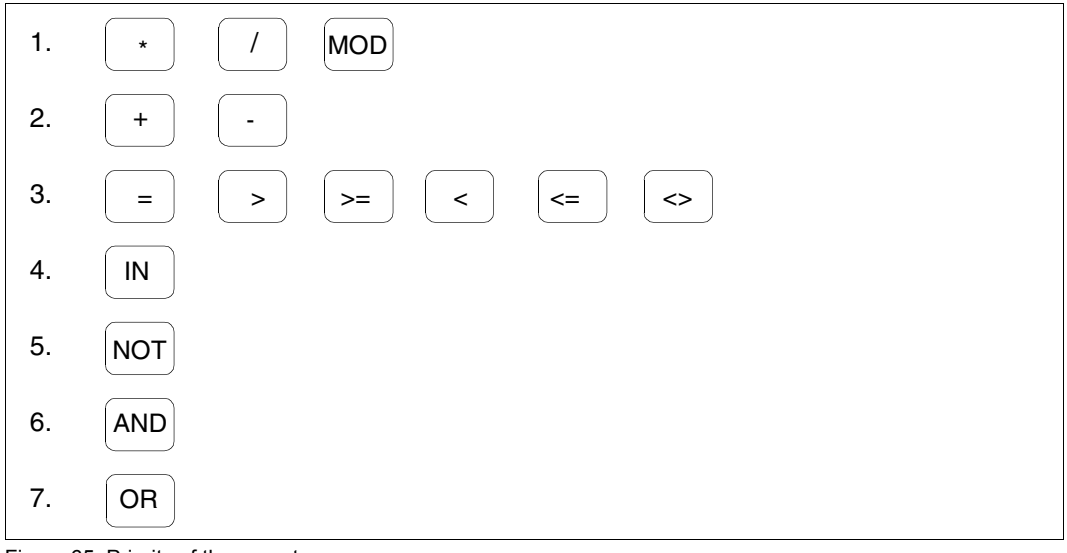


Figure 65: Priority of the operators

5.7.3.3 Data types

PRODAMP recognizes the following three data types:

- the **numeric data type** with a length of 1 to 4 bytes,
- the **string** with a length of 1 to 133 bytes and
- the **bit pattern** with a length of 1 to 4 bytes.

The **numeric data type** with a length of 4 bytes is the only numeric data type used for variables and constants and is simply referred to as a numeric data type. It can hold an integer in the range -2147483648 to +2147483647. The integer -2147483648 is accepted as a literal only in the hexadecimal form (X'80000000').

This data type is also used for addressing diagnostic data. Note that only the first 31 bits are used to construct addresses of /390 objects; all 32 bits are used for x86 objects.

The addressing mode is an HSI-dependent constant for PRODAMP.

The numeric data types with a length of 1 to 3 occur with symbols only and constitute the numeric interpretation of fields 1 to 3 bytes in length. For numeric data types with a length of 2, the contents are interpreted as a signed number (halfword arithmetic).

Numeric literals may be specified in decimal or hexadecimal format.

Examples

```
123
-1234
X'0AFFE'
```

The **string** is used to represent character strings enclosed in apostrophes. Two consecutive apostrophes are used to represent a single apostrophe within a string. The maximum length of a string in PRODAMP is 133 characters. If a string contains non-printable characters, it can be declared as a hexadecimal string. In this case, the Pascal convention of using the string #' as the left-hand delimiter and the apostrophe ' as the right-hand delimiter is used.

Examples

```
'1'
'Long string literal',
'Value: X''01''',
#'C100C600C600C500';
```

Bit patterns are data items with a length of 1 to 4 bytes in which each individual bit can be addressed. Bit pattern literals are defined either in binary (up to one byte long) or in hexadecimal form. The notation for binary definition is the same as for Assembler. For hexadecimal notation, the string P' is used as the left-hand delimiter and the apostrophe ' as the right-hand delimiter in order to avoid confusion with hexadecimal numeric literals.

PRODAMP also has the symbolic bit pattern constants TRUE and FALSE.

Examples

```
B'10101001'  corresponds to P'A9'
TRUE         corresponds to P'01' or B'1'
FALSE        corresponds to P'00' or B'0'
```

Compatibility of data types

The following table shows the permissible type mixtures for assignments (see page 248) and comparison operations.

Right → / Left ↓	Undefined	Numeric	Bit pattern	String	
Undefined	Arithmetic	Arithmetic	Bit pattern	String	*)
Numeric	Arithmetic	Arithmetic	Arithmetic	-----	
Bit pattern	Bit pattern	-----	Bit pattern	-----	
String	String	-----	-----	String	

Table 9: Permissible type mixtures for assignments and comparison operations

*) The entries in this line apply only to assignments. For comparisons, the operands may have to be regrouped to ensure that the left-hand operand is not “undefined”.

The basic rule is: the type of an expression is determined by the type of the first term.

For assignments, “left” denotes the variable to the left of the assignment operator and “right” denotes the expression to the right of this operator. The field at the intersection point shows whether the assignment is permitted or prohibited and specifies the result type (the latter is interesting only if the type was previously undefined).

For comparison operations, “left” denotes the expression to the left of the comparison operator and “right” denotes the expression to the right of this operator. The field at the intersection point shows whether the comparison is permitted and how it should be interpreted.
(For an arithmetical comparison, for instance, the operator “IN” is prohibited.)

In expressions, types may be mixed in only two ways:

<numeric><operator><bit pattern> (results in a numeric) and
<string><operator><numeric> (results in a string).

In the second case, only the operator “*” is accepted (for string replication). Expressions in which the first term is undefined receive the type “numeric”.

Symbols are always “undefined” because the symbol file is accessed only when the procedure is executed. This can lead to undesired results, as shown in the following example.

Example

Let us assume you are interested in the rightmost bit in field EXVTAUDI. You thus want to mask out the high-order bits and write:

```
MY_BITS:=.EXVTAUDI-P'FE';
IF MY_BITS=P'01' THEN
```

In this case, EXVTAUDI is set to arithmetic, the bit pattern literal is also interpreted as arithmetic and the result in MY_BITS is either the arithmetical value -253 (X'FFFFFF03') if the bit is set or -254 if it is not set. The following IF statement is accepted by the compiler, because it regards MY_BITS as arithmetic and therefore converts the bit pattern to arithmetic again (see table 9 on page 233). However, this was not what you wanted.

To get what you actually wanted you have to enter

```
MY_BITS:=P'0'+.EXVTAUDI-P'FE';
IF MY_BITS=P'01' THEN
```

Alternatively, you can initialize variables to ensure that the correct operations are generated when symbols are used.

In order to access the diagnosis object via symbols whose types are not known at compilation time, it is thus necessary to consider all the implications if you want to avoid unpleasant surprises. Systematically assigning all diagnostic data to initialized variables is one way of avoiding such problems.

The following table shows how the data types contained in the symbol files in the form of LSD codes (Assembler data types) are converted into the data types used by PRODAMP.

LSD code	Ass. type	Data length	PRODAMP type	Data description
00	C	n	STRING	Character
01	Z	n		Unpacked decimal
03	E	4		Floating-point, single-precision
04	D	8		Floating-point, double-precision
05	P	n		Packed decimal
0F		n		Machine instruction
10		n		CSECT, COM, DSECT, XDSEC
06	H	2	NUMERIC	Signed binary
07	F	4	NUMERIC	Signed binary
0A	Y	2	PATTERN	Unsigned binary
	A	4	NUMERIC	Unsigned binary
0A	X	1 - 3	PATTERN	Unsigned binary
		4	NUMERIC	
		otherwise	STRING	

Table 10: Conversion of Assembler data types to PRODAMP data types

5.7.3.4 Symbols

Symbols are used to access the object being diagnosed (dump file or the active system) or the metadata of the DAMP program. Each symbol has a name beginning with a period, a relative address, a type and a length.

Symbols cannot be named freely, but must

– be known to DAMP from the symbol file,
– be created by means of the ARRANGE statement within the PRODAMP procedure or
– be defined internally, i.e. by the PRODAMP programming language.

The name, relative address, type and length of a symbol are stored in the DAMP symbol files or in the private symbol files assigned by means of ADD-SYMBOLS. The relative address always refers to the beginning of the structure (DSECT) which contains the appropriate symbol. This means that a data structure in the diagnosis object can be accessed via a symbol only if the base address of the structure concerned is specified. This fact is taken into account in the syntax of a symbol.

Examples

```
.ETCBTFT
```

“.ETCBTFT” is identified as a symbol by the initial period. The base address does not need to be specified explicitly in this case because the TCB belongs to the structures which DAMP can localize automatically (other such structures are the JCB, UVMT, SVMT and EXVT).

```
A_FCB.ID1FILE
```

“A_FCB” is a variable which contains the base address of the structure concerned - in this case, a TU-FCB. “ID1FILE” is a field name in the DSECT “ID1FCB”.

```
PTR.NKLCB_MDL.COPY_PARAMETER.USER_ALLOCATION.WAIT_FACTOR
```

This example shows symbols in substructures. These are specified in terms of the nesting structure of the substructure or by calling the standard PRODAMP procedure REFERENCE.
It is only necessary to specify the DSECT (NKLCB_MDL) if the first symbol (COPY_PARAMETER) is not unique among all the DSECTS contained in the symbol file.

The following names are reserved as identifiers in PRODAMP and cannot be used for variables:

ABS-ADDRESSING	ADDRESS	ALET	AND	ARRANGE
ASEL	COMMAND	CPU	CURRENT	DEC_BINARY
DEC_STRING	DMP_#REFRESH	DO	DSECT	DUMP_MEMORY
ELSE	ELSIF	END	ENTER_MODULE	EXTRACT
FALSE	FOLLOW	HEX_BINARY	HEX_STRING	HSA
IF	IN	INFIELDS	INSERT	INTERRUPT
ITN	LAYOUT	LENGTH	LIST	LOCATION
MESSAGE	MOD	NAME	NEW_TASK	NEXT_WINDOW
NOT	NUMBER	NUMERIC	OFF	OFFSET
ON	OPC_TABLE	OR	OUTPUT	PARAMETER
PATTERN	PCB	PCK_BINARY	READ	READ_WINDOW
REAL	REFERENCE	RELATIVE	RETURN	SET_HEADER
SPID	STRING	SVC_TABLE	THEN	TID
TRACE	TRUE	TSN	TYPE	UNDEFINED
UNSIGNED_OFF	UNSIGNED_ON	VIRTUAL	WHILE	WINDOW
WRITE				

The following names are reserved as symbol identifiers in conjunction with CURRENT and INFIELDS in PRODAMP. If they are not included in the list above, they can be used as variables, but not as symbol names.

ADDRESS	ALET	ASEL	ASID	ATYPE
COMMAND	CONFIGURATION	CPU	CSMA	DTYPE
DUMPTIME	ERROR	FILENAME	HSA	ITN
LAYOUT	LENGTH	LEVEL	MARK1	MARK2
MARK3	MARK4	MARK5	MARK6	PARAMETER
PCB	PTYPE	RELATIVE	SEGMENT	SPID
STACK	SYMBOL	TIME	TID	TSN
VERSION	WNDNO	WNDTSK		

PRODAMP procedures with these names cannot be called as user subroutines.

All identifiers that begin with DMP_ are reserved for future PRODAMP extensions and should therefore not be used in user programs.

5.7.3.5 Variable

A variable is an object to which values can be passed during program execution. It belongs to one of the data types and can be both written to and read.

Variables can be fixed-length strings (up to 133 characters), numeric values (4 bytes) or bit patterns (1 to 4 bytes).
During compilation, the data type of a variable is defined by statically the first assignment, and by the defined data type on the right of the assignment. See table 9 on page 233 for more details.
During runtime, any attempts to read a variable that has not yet been dynamically defined will result in a runtime error and in termination of the PRODAMP program.

5.7.3.6 Expressions

Expressions are calculation rules which, after evaluation, return a value. They are formed by combining operands with operators (see section “Syntax diagrams” on page 312). The meanings of the operator symbols are shown in the following table:

	Numeric	**String**	**Bit pattern**
+	Addition	Concatenation	Set union
-	Subtraction		Set difference
*	Multiplication	Replication *)	Set intersection
/	Division		
MOD	Modulo operation		

*) The second operand must in this case be numeric.

Table 11: Meanings of the operators

In PRODAMP, the operators are thus ambiguous, i.e. they are interpreted differently for different operand types.

Examples

```
33 + 16                 Result: 49
X'10' * 4               Result: 64
'System' + ' crash'     Result: 'System crash'
#'C1' * 5               Result: 'AAAAA'
B'1001' + B'11'         Result: B'1011' or P'0B'
P'1A' * B'1001'         Result: B'1000' or P'08'
```

The modulo operation returns the (integer) remainder left over by a division.

Examples

```
29 MOD  7      produces 1
35 MOD 11      produces 2
```

The MOD operand can also be used, for example, to convert addresses “manually” to 24-bit addresses.

Example

```
X'887C0A0E' MOD X'01000000' Result: X'007C0A0E'
```

Expressions can be enclosed in parentheses in the normal manner and can be combined as required (providing the types are compatible).

Examples

```
X * Y + 3 * ( A – B ) + X'ABC'
A_FCB.ID2IND1 – P'80'            All bits from ID2IND1 except ID2DUMMY
NUM – ( NUM / 16 ) * 16
('a'+'b'*2) * 4                  Result: 'abbabbabbabb'
```

5.7.3.7 Statements

A PRODAMP procedure consists of a number of statements. Each statement is terminated by a semicolon. Since there are no declarations in PRODAMP, each statement generates code which can be interpreted only when the procedure is executed.

PRODAMP incorporates the following statements:

Statement name	Function
ARRANGE	Define symbol attributes
FOLLOW	Monitor variables
IF	Issue conditional statements
INTERRUPT	Interrupt procedures
RETURN	Leave a procedure
TRACE	Control tracing
WHILE	Form program loops
Procedure call	Call a procedure
Assignment	Assign a value

Table 12: Overview of PRODAMP statements

ARRANGE
Define symbol attributes

The ARRANGE statement can be used to declare the names, lengths, relative addresses and types of symbols.
ARRANGE WINDOW can be used to specify that a particular dump window is to be displayed as the top window on screen. In addition, it is possible to specify that values from the PRODAMP procedure are to be entered in the input fields in the window. The keywords and their assignments are given in figure 66.

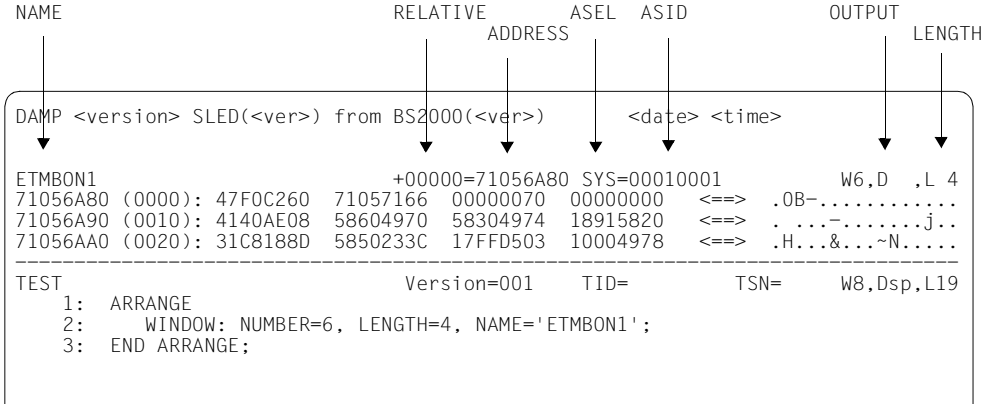


Figure 66: Header line of a dump window.

ARRANGE WINDOW

ARRANGE WINDOW moves the specified window to the top of the window chain in DAMP. After several ARRANGE WINDOW statements for different windows, followed by the screen display in each case, the last window which was defined thus appears as the topmost window on the screen. The other windows are displayed in the reverse order, provided there is enough space to accommodate their defined sizes.

ARRANGE WINDOW specifications are not permitted for the help window (W1), PROC window(s) and other special windows and are rejected. The parameters TID, TSN and LENGTH may be declared for the status window (W2) and the stack window (W3); for W3, also the PCB with which a PCB can be selected from the PCB chain of the task via its sequence number.

ARRANGE WINDOW acts just like an input in the header line of a diagnostic window. Each possible input in the header has a corresponding keyword in the ARRANGE statement. The only exception to this is the parameter NUMBER, via which the desired window number is specified.

ARRANGE

Data structures which are not stored in a symbol file can be made available on the fly by means of the ARRANGE statement. ARRANGE is also always necessary when unstructured dump data is to be addressed (see “Example 5 for ARRANGE” on page 242).

The symbols (re)defined via ARRANGE are available only within the PRODAMP procedure: they are not placed in the binary symbol tree of DAMP. This means that these changes have no effect when a window is overlaid with a DSECT.

New symbols must be defined completely, i.e. LENGTH, TYPE and RELATIVE must be specified explicitly. Otherwise, PRODAMP attempts to fill out the missing specifications with information from the symbol file; if the symbol cannot be found, PRODAMP aborts execution of the procedure.

Conversely, existing characteristics can be reset explicitly by specifying the keyword UNDEFINED for them. The next time this symbol is accessed, the missing characteristic is then filled out with information from the symbol file.

Example 1 for ARRANGE WINDOW

```
ARRANGE WINDOW:
NUMBER=5, ADDRESS=0, LENGTH=8, TID=X'ABC', OUTPUT='ASS' ;
END ARRANGE ;
```

Window 5 is to display the user memory of task ABC, starting at address 0. The output is to be disassembled (OUTPUT='ASS'). Furthermore, the size of the window is to be 8 lines.

Example 2 for ARRANGE WINDOW

```
ARRANGE WINDOW:
NUMBER=7, NAME='IDMFILE', DSECT = 'IDMTFT', OUTPUT='CBM',
ADDRESS=TFT_AD;
END ARRANGE ;
```

Window 7 is to display a memory area starting at address TFT_AD. The area is to be edited symbolically in the format of DSECT IDMTFT. The DSECT is positioned such that the DSECT IDMFILE is at the base address TFT_AD and the field IDMFILE is in the first line of the window. The window start address is TFT_AD + offset(IDMFILE).

Example 3 for ARRANGE

```
ARRANGE
.ESTK#ICL : TYPE = STRING ;
END ARRANGE ;

IF 'P' = STK1.ESTK#ICL THEN ...
```

The field ESTK#ICL in DSECT ESTK is redefined. It is defined as DS XL1, which means that it is interpreted as a bit pattern, but it contains a letter. Assignment of the type STRING makes it possible to interpret this field as a letter in subsequent statements.

Example 4 for ARRANGE

```
ARRANGE
.STDHD : RELATIVE = 0, LENGTH = 8, TYPE = STRING ;
.PAR_1 : RELATIVE = 8, LENGTH = 4, TYPE = NUMERIC ;
.PAR_2 : RELATIVE =12, LENGTH = 1, TYPE = STRING ;
.PAR_3 : RELATIVE =13, LENGTH = 1, TYPE = PATTERN;
END ARRANGE ;

P_LATTE := .... ;
FRST_PAR := P_LATTE.PAR_1 ; ....
```

Within PRODAMP, a DSECT is constructed, e.g. for a newly developed interface whose parameter list has not yet been passed officially. The fields of the parameter list can now be addressed symbolically using the newly defined names; the related base address must, of course, be specified as well. A similar method is used to pass parameters between PRODAMP procedures.

Example 5 for ARRANGE

```
ARRANGE
.BYTE : TYPE=NUMERIC,LENGTH=1,RELATIVE=0;
END ARRANGE;
INT_AD := STK.ESTK#ICR;           "Address of the paging error"
OPCODE := INT_AD.BYTE;            "Machine instruction at this point"
IF OPCODE = X'D2' THEN ......
```

ARRANGE can be used to address unstructured data of the diagnosis object (such as coding, for example).

Example 6 for ARRANGE

```
ARRANGE
 .HALFWORD: OFFSET = 0, LENGTH = 2, TYPE = NUMERIC;
END ARRANGE
WORD := PTR.HALFWORD
```

If a field is defined as NUMERIC with LENGTH=2, then special care must be taken to ensure that the sign is taken into account when the contents of the file are assigned to a variable.
If HALFWORD contains, for instance, the sequence of characters C'AB' at the address indicated by PTR, this does not correspond to the numeric value X'C1C2' = 49602 after the field has been defined. Instead, it corresponds to the value X'FFFFC1C2' = -15943, which is the value subsequently assigned to the variable WORD. If the sign for the contents is to be ignored, the field must be defined with TYPE=PATTERN. The general rule for TYPE=NUMERIC is that the sign is taken into account if the length is an even value and the value transferred is always positive if the length is an odd value.

Example 7 for ARRANGE

```
ARRANGE
 .WAIT_FACTOR: LENGTH = 1, TYPE = STRING,
               REFERENCE = .NKLCB_MDL.COPY_PARAMETER.USER_ALLOCATION
END ARRANGE;
```

The elements of a substructure can also be redefined using ARRANGE. To do this, you must use the name REFERENCE in order to localize the symbol within a reference chain.

FOLLOW
Monitor variable

The FOLLOW statement is used to monitor variables. This is possible only if the variable has already been declared, i.e. initialized.

Following execution of the FOLLOW statement, all assignments to the specified variable are logged in an EDT area (by default, area 8). The output includes the number of the line in which the variable is assigned, the name of the variable, and its value after assignment.

In addition, every change to the CURRENT.ERROR value is logged.

If the EDT area is changed during procedure execution by means of @WRITE ("@PROC XX"), the output for variable monitoring is likewise directed to the new EDT area.

FOLLOW is very useful for searching for errors in PRODAMP procedures.

Variable monitoring can be activated but not deactivated, i.e. it is not terminated until the end of the procedure or until leaving the procedure by means of RETURN.

Example for FOLLOW

The procedure

```
1)  S := ' '*8;
2)  STR := '12XY34';
3)  NUM := ' ';
4)  I := 0;
5)  A := 0; FOLLOW A;
6)  WHILE I <= 5 DO
7)    EXTRACT ( NUM,STR,I );
8)    INSERT ( NUM,S,3 );
9)    A := DEC_BINARY ( S );
10)    I := I + 1;
11)  END WHILE;
```

generates the following log:

```
TEST (0) %STMT 9: 'A'              <- 1 ( X'00000001' )
TEST (0) %STMT 9: 'A'              <- 2 ( X'00000002' )
TEST (0) %STMT 9: 'A'              <- 1 ( X'00000001' )
TEST (0) %STMT 10: 'CURRENT.ERROR' <- 4
TEST (0) %STMT 9: 'A'              <- 1 ( X'00000001' )
TEST (0) %STMT 9: 'A'              <- 3 ( X'00000003' )
TEST (0) %STMT 10: 'CURRENT.ERROR' <- 0
TEST (0) %STMT 9: 'A'              <- 4 ( X'00000004' )
```

Output of the change to the CURRENT.ERROR value caused by a standard procedure comes after the statement that caused it. The example also shows that DEC_BINARY returns an undefined value if the string is not a decimal number.

IF
Issue conditional statements

The IF statement can be used to make the execution of other statements contingent on specified conditions. The condition to be fulfilled is defined by combining several expressions with the aid of **logical operators** (see page 230).

Example 1 for IF

```
IF PCB.ESTKGR15 <> 0 THEN
  RC := PCB.ESTKGR15 MOD 256 ;
  MESSAGE ( 'Returncode '+HEX_STRING(RC)+' for $REQM.' );
END IF;
```

If the contents of field ESTKGR15 are not equal to zero, i.e. if register 15 contains a return code, the message given after it is displayed on the screen.

Example 2 for IF

```
IF 'TSOS ' = .EJTPUSR THEN
  IF 'HELGA' + ' '*49 = .EJTPXPRG THEN
    DANGER := 100 ;
  END IF ;
ELSIF 'SERVICE ' = .EJTPUSR THEN
  DANGER := 180 ;
ELSE
  DANGER := 0 ;
END IF;
```

The user and the program used are queried in fields EJTPUSR and EJTPXPRG. The results determine what value is set for the DANGER variable.

Example 3 for IF

```
EVIPLFLG := B'00000100' ; 'SYSTEM LOADING COMPLETED'
EVCL2REQ := P'02' ; 'CLASS 2 MEMORY REQUESTED'
IF EVIPLFLG + EVCL2REQ IN .EVSVMIND THEN
```

Two fields are defined. A check is made as to whether the contents of these fields can be found in the EVSVMIND field.

Example 3 is interesting for another reason: it deals with concrete equates from the DSECT SVMT. Since DAMP does not store the equates in the symbol file, the corresponding bit patterns have to be defined in PRODAMP.

INTERRUPT
Interrupt procedure

The INTERRUPT statement enables the use to interrupt a procedure and branch to the DAMP program, where it is then possible to enter any DAMP statements. Control is returned to the procedure by means of the RESUME-PRODAMP-PROGRAM statement.

INTERRUPT is useful, for example, when a chain of data structures has to be examined sequentially. For instance, if the INTERRUPT statement is contained in a loop that localizes the data structures one after the other and assigns them to a screen window, the user can, with the added aid of the RESUME-PRODAMP-PROGRAM statement, as it were hop from one link of the chain to the next.

Example for INTERRUPT

```
INTERRUPT WINDOW=4
```

The parameter WINDOW= causes the entire DAMP screen to be refreshed. If several outputs are sent one after the other to the same window by means of INTERRUPT, this can often be somewhat irritating. The refresh operation can be avoided by using INTERRUPT without an explicit window specification; in this case, only the modified fields on the screen are rewritten.

RETURN
Leave procedure prematurely

Issuing the RETURN statement in a PRODAMP procedure returns control to the point where the procedure was called. This may be either another procedure or the DAMP dialog.

RETURN is also generated implicitly by the compiler at the end of each procedure.

RETURN WINDOW=
Terminate procedure prematurely

In contrast to a simple RETURN, the statement RETURN WINDOW=<window-number> not only exits the current procedure, but aborts the entire call hierarchy. In addition, <window-number> specifies the number of the diagnosis window that is to be displayed in the dialog at the topmost position on the DAMP screen after aborting the procedure.

Example for RETURN

```
RETURN WINDOW = CURRENT.WNDNO;
```

This statement fully exits the procedure and sets the diagnosis window to the one specified in the CURRENT.WNDNO field.

TRACE
Control tracing

The statements TRACE ON and TRACE OFF activate and deactivate tracing. All source line numbers encountered between TRACE ON and TRACE OFF are logged in an EDT area (unless otherwise specified, area 8). Both statements can be entered anywhere within a procedure.

If, in the course of procedure execution, the EDT procedure area is changed by means of WRITE (“@PROC XX), the trace is also output to the new area.

Activation and deactivation of the trace are contingent upon the dynamic statement sequence.

TRACE is extremely useful for searching for errors in PRODAMP procedures.

Example for TRACE

Execution of the procedure PROCNAME:

```
1) IF CURRENT.LEVEL < 4 THEN
2)    TRACE ON;
3)    I := 1234;
4)    PROCNAME;
5)    I := 5678;
6) END IF;
```

causes the following line numbers to be logged:

```
PROCNAME (0) %STMT 3
PROCNAME (0) %STMT 4
PROCNAME (1) %STMT 3
PROCNAME (1) %STMT 4
PROCNAME (2) %STMT 3
PROCNAME (2) %STMT 4
PROCNAME (3) %STMT 3
PROCNAME (3) %STMT 4
PROCNAME (3) %STMT 5
PROCNAME (3) %STMT 6
PROCNAME (3) %STMT 7
PROCNAME (2) %STMT 5
PROCNAME (2) %STMT 6
PROCNAME (2) %STMT 7
PROCNAME (1) %STMT 5
PROCNAME (1) %STMT 6
PROCNAME (1) %STMT 7
PROCNAME (0) %STMT 5
PROCNAME (0) %STMT 6
PROCNAME (0) %STMT 7
```

The procedure name is output first, followed by the value of CURRENT.LEVEL in parentheses, to enable the subprogram levels of the procedure to be distinguished. Then the line number of the processed statement is output. At the end of a procedure PRODAMP always implicitly generates a RETURN statement.

As a result, the log will contain STMT 7, which is not contained in the above procedure.

If tracing is activated, any further TRACE ON statements will be ignored. The first TRACE OFF statement deactivates tracing.

WHILE
Form program loops

The WHILE statement can be used to form loops.

Example for WHILE

```
WHILE ADDR <= MAX DO
   IF ADDR.TABVAL = BAD_VAL THEN
     RETURN ;
   END IF ;
   ADDR := ADDR + TAB_LEN ;
END WHILE ;
```

As long as the variable ADDR does not become greater than the variable MAX, the subsequent statement sequence will be processed repeatedly.

Assignment

The simplest form of statement is the **assignment**. The assignment operator is the character combination ":=". This operator must be preceded by the name of a variable and followed by an expression, i.e. the name of a variable, a symbol, a literal, a function call or an arithmetic expression.

The data type of a variable is defined by the first static assignment (i.e. in the sequence of the source) to this variable, as shown in table 9 on page 233. The data type on the right is determined and transferred, taking into consideration any existing type assignments. If the data type on the right is undefined, it is set to numeric. A data type is undefined if, for example, the first operand is a symbol; all symbols, including those defined with the PRODAMP statement ARRANGE, have the value "undefined" during compilation.

Examples

```
A_FCB  := X'ABC';
FNAM   := A_FCB.ID1FILE;
TEXT   := 'The specified FCB does not exist.';
XY     := FALSE;
```

In the above examples (assuming that these are statically the first assignments to the variables concerned), A_FCB will therefore be defined as numeric, TEXT as a string with a length of 33, and XY as a bit pattern.

Since the type for the symbol ID1FILE cannot be determined at compilation time, FNAM receives the default type “numeric”. (Otherwise, the symbol file which is to be used later for procedure execution would have to be assigned during compilation, even if the procedure is intended for use with a completely different version of BS2000.) For this reason, it is best to declare variables by assigning them an initial value in cases where (possibly non-numeric) data from the diagnosis object is to be assigned to them later with the aid of a symbol.

Examples

```
A_FCB  := X'ABC' ;
FNAM   := ' ' * 54 ;
FNAM   := A_FCB.ID1FILE ;
```

The right side of an assignment may also consist of function calls (for standard functions) and expressions.

If a string is assigned to a string of another length that has already been initialized, the source string is truncated in the case of a shorter target string, or padded with blanks in the case of a longer target string.

5.7.3.8 Pseudo-structures

The pseudo-bases CURRENT and PARAMETER can be used to access operating data of the DAMP program.

The pseudo-base CURRENT

Diagnostic algorithms require not only the data of the object being diagnosed, but sometimes also the “operating data” of the DAMP program. This can be achieved by means of the pseudo-base “CURRENT.item”. The available items are explained below.

With the exception of CURRENT.ALET, CURRENT.ATYPE, CURRENT.ERROR, CURRENT.SPID and CURRENT.SEGMENT, the fields cannot be overwritten. Any attempt to assign a value to a read-only field will result in a compiler error.

CURRENT.ALET

When accessing data spaces (see CURRENT.ATYPE), CURRENT.ALET must be set to the required value for ALET (4-digit numeric value). The corresponding TID must be set using the standard procedure NEW_TASK (see page 277).

CURRENT.ATYPE

Unless otherwise specified, addresses are interpreted as virtual addresses in PRODAMP. However, it is also possible to address real memory, the hardware system area (HSA) or data spaces. This is done with the aid of the pseudo-symbol CURRENT.ATYPE and the pseudo-constants VIRTUAL, REAL, HSA, ABS_ADDRESSING, ALET and SPID, which can be assigned to this pseudo-symbol. Such an assignment defines which memory is to be addressed by the subsequent statements.

By assigning ABS_ADDRESSING to the pseudo symbol CURRENT.ATYPE, absolute memory can be addressed in the complete VM2000 SLED, e.g. to analyze the hypervisor.

Example 1 for CURRENT.ATYPE: Addressing the HSA

```
HSA_START  := CURRENT.HSA;
TEST_VALUE := 0;
ARRANGE
  .TEST_SYMBOL : TYPE=NUMERIC,LENGTH=1,RELATIVE=0;
END ARRANGE ;
CURRENT.ATYPE := HSA;
TEST_VALUE := HSA_START.TEST_SYMBOL;
IF CURRENT.ERROR <> 0 THEN
  MESSAGE ( 'Hardware system area is not addressable !!' );
ELSE
  ....
END IF;
CURRENT.ATYPE := VIRTUAL;
```

However, certain restrictions must be noted: the selected memory type applies only when addressing memory areas with the aid of symbols and is effective only locally within the current procedure. If called procedures are also to address real memory, the HSA, absolute memory or data spaces, then the memory type must also be set locally in these procedures. The only exception to this rule is the predefined procedure DUMP_MEMORY (see page 269), which permits real memory, HSA memory, absolute memory or areas from data spaces to be output directly to a list.

If ALET and SPID are specified, the required values must be set before they are accessed using CURRENT. This is demonstrated in the following example:

Example 2 for CURRENT.ATYPE

```
ARRANGE
  .TEST.SYMBOL : TYPE = NUMERIC, LENGTH = 4, OFFSET = 0;
END ARRANGE;
CURRENT.ATYPE := ALET;
CURRENT.ALET := X'01010003';
PTR := 0;
OUT := PTR.TEST_SYMBOL;
IF CURRENT.ERROR <> 0 THEN
  MESSAGE ( 'Access error, ALET = '
          + HEX_STRING(CURRENT.ALET,8) + ', TID = '
          + HEX_STRING(CURRENT.TID,8) );
ELSE
...
END IF;
CURRENT.ATYPE := VIRTUAL;
```

If ALET is specified, the TID currently set is always used for access purposes. The TID can be reset as required using the standard procedure NEW_TASK (see page 277).

Furthermore, symbols which are localized automatically are **always** addressed virtually. Consequently, if you want to address real memory, absolute memory or the HSA, you must always specify a base address (as in the example above).

More than a simple assignment is also possible. Since the pseudo-symbol CURRENT.ATYPE has, internally, the type numeric, you do not have to explicitly specify VIRTUAL, REAL, HSA, ABS_ADDRESSING, ALET or SPID. Instead, you could perform calculations with the value (although this is rather pointless), transfer the current value to another variable or, for example, pass the memory type as a parameter to another procedure. If an invalid value is assigned to CURRENT.ATYPE, this is not detected until runtime and causes the procedure to be aborted. The validity of the values is not checked at compilation time.

Example 3 for CURRENT.ATYPE

```
CURRENT.ATYPE:=35
```

This assignment is syntactically correct.

Example 4 for CURRENT.ATYPE transferring parameters

A procedure which returns the contents of the real address 0 as a printable string is called.

```
CALLER:

  VALUE := '        ';
  PRINTABLE_VALUE ( 0, REAL, VALUE );

PROCEDURE PRINTABLE=VALUE :

  LOC := 0;
  ARRANGE
    .ADDR     : TYPE=NUMERIC, LENGTH=4, RELATIV=0;  'PARAMETER 1'
    .MEMORY   : TYPE=NUMERIC, LENGTH=4, RELATIV=4;  'PARAMETER 2'
    .TARGET   : TYPE=STRING,  LENGTH=8, RELATIV=8;  'PARAMETER 3'
    .CONTENTS : TYPE=NUMERIC, LENGTH=4, RELATIV=0;
  END ARRANGE
  CURRENT.ATYPE := PARAMETER.MEMORY;
  LOC := PARAMETER.ADDR;
  PARAMETER.TARGET := HEX_STRING ( LOC.CONTENTS, 8 );
  RETURN;
```

CURRENT.CPU

CURRENT.CPU contains the name of the CPU from the diagnosis object in the form of a string with a length of 8. If a file is opened as a PAM file, the CPU of the current system is displayed.

CURRENT.CONFIGURATION

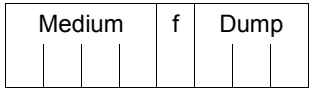
CURRENT.CONFIGURATION contains the designation of the hardware configuration from the diagnosis object in the form of a string with a length of 21 (e.g.: 7.500- S210-60). The output corresponds to the designation in the /SHOW-SYSTEM-INFORMATION command. The configuration of the active system is output for a file opened as a PAM file.

CURRENT.CSMA

CURRENT.CSMA contains the (absolute) start address of the common shadow memory area (CSMA), i.e. a 4-digit numeric value.

CURRENT.DTYPE

CURRENT.DTYPE contains a 1-byte (8-bit) numeric value describing the opened medium. This value has the following format:



The following meanings apply:

Medium = 0 (B'0000') : Medium not defined
Medium = 1 (B'0001') : System
Medium = 2 (B'0010') : Object can be selected
Medium = 3 (B'0011') : Dump
Medium = 4 (B'0100') : PAM file
Medium = 5 (B'0101') : Self-loader

The Medium = 2 case can only occur in the dialog if the dump file contains multiple objects and if no subsequent selection (e.g. complete VM2000 SLED, SLED from a SLED) has been made in the INF screen as yet. In such cases, the program should therefore be aborted and a selection made in the INF screen. If no selection was made in the INF screen, access to the object (only a complete VM2000 SLED) with PRODAMP is only possible in the absolute addressing mode (CURRENT.ATYPE = ABS_ADDRESSING).

If Medium = 3 (0011), f and dump are both set. The following meanings apply:

Dump = 0 (B'000') : SLED
Dump = 1 (B'001') : System dump
Dump = 2 (B'010') : User dump

If f is set to 1 (only for a SLED or user dump), the SLED is a snap file or the user dump is an area dump.

Example for CURRENT.DTYPE

The following query could be issued if you wish to check whether the opened medium is a user dump:

```
IF  CURRENT.DTYPE / 16 = 3  AND
    CURRENT.DTYPE MOD 8 = 2
THEN
```

CURRENT.DUMPTIME

CURRENT.DUMPTIME contains the date and time of the dump file in the format of a string 19 characters long: yyyy-mm-dd hh:mm:ss. The current date and time is supplied for the diagnosis of the active system.

CURRENT.ERROR

If the program detects an inconsistency which can be handled internally, i.e. which does not force the PRODAMP procedure to be aborted, then CURRENT.ERROR is set to a value other than 0. For example, if a requested page is missing from a dump file, this does not justify abortion of the procedure. But if, in contrast, a specified symbol cannot be found in the associated symbol file, PRODAMP aborts the procedure, as this problem can usually be traced to a typing error in the source code of the PRODAMP procedure, which will have to be corrected.

Theoretically, each access to a datum of the diagnosis object can result in an error. This should be checked by inspecting the contents of CURRENT.ERROR, since further execution of the PRODAMP procedure will be unpredictable if an error has occurred. To avoid impairing performance and to keep the procedure as simple as possible, this check can be restricted to a one-off query after the first access, albeit only if several objects within the same data structure are accessed one after the other and if you are sure that the data structure does not exceed the memory page.

The error code that may possibly be stored in CURRENT.ERROR is irrelevant for the diagnosis. You should therefore compare CURRENT.ERROR only against 0.

CURRENT.ERROR can be explicitly reset by assigning it the value 0 (CURRENT.ERROR := 0). This is not required in many cases, however, since PRODAMP automatically resets the CURRENT.ERROR in the following situations:

– On entering a procedure (CURRENT.ERROR is maintained on a local procedure basis).

– Following a successful read access on the data of the diagnosis object or the operating data of DAMP by means of a symbol (see section “Symbols” on page 236).

  Example

  X:=.ETCBTID, Z:= CURRENT.DTYPE, Y:=PARAMETER .P1

  Exception: read access on CURRENT.ERROR.

– On successfully using one of the standard functions of DAMP.
  Exception: the call to PATTERN does not change CURRENT.ERROR.

– On successfully executing the standard procedures NEW_TASK or READ.

CURRENT.ERROR is not reset by any of the other standard procedures and assignments within PRODAMP.

Example for CURRENT.ERROR

```
TEST := P2_FCB.ID2CFLID ; "PSEUDO-HARDVALIDATION"
IF CURRENT.ERROR = 0 THEN
  "ACCESS POSSIBLE WITHOUT RISC"
  IF P2_FCB.ID2IND1 = ID2PRIVC THEN
    IF P2_FCB.ID2LOCK = ID2OUTL THEN
    "DO SOMETHING"
    END IF ;
  END IF ;
END IF ;
```

CURRENT.FILENAME

CURRENT.FILENAME contains the name of the currently open diagnosis object in the form of a string with a length of 54. If the diagnosis object is the active system, the string contains only blanks.

CURRENT.HSA

CURRENT.HSA contains, for SLED files, the (absolute) start address of the hardware system area. It is thus a numerical value with a length of 4. Since only SLED files have a hardware system area, CURRENT.HSA has the value 0 for other dump files.

CURRENT.ITN

See CURRENT.TID.

CURRENT.LEVEL

The pseudo-symbol CURRENT.LEVEL contains a numerical value with a length of 1 which shows the nesting depth of subroutine calls. A procedure in which RETURN would return control to the DAMP environment has CURRENT.LEVEL 0; a procedure called by this procedure has CURRENT LEVEL 1, and so on.

CURRENT.PCB

The pseudo-symbol CURRENT.PCB contains the address of the current PCB, which is set in diagnostic window 3. It is thus a numeric value with a length of 4. If no PCB has been set, CURRENT.PCB contains the value -1 (X'FFFFFFFF').

CURRENT.PTYPE

CURRENT.PTYPE contains a 1-byte numerical value indicating the DAMP execution mode.
The following values are possible:

CURRENT.PTYPE = 0 (B'00000000') : DAMP is running in interactive mode
CURRENT.PTYPE = 1 (B'00000001') : DAMP is running in procedure mode
CURRENT.PTYPE = 2 (B'00000010') : DAMP is running in batch mode

CURRENT.SEGMENT

CURRENT.SEGMENT is required when the diagnosis object is to be accessed with very large addresses (i.e. with an address width of more than 32 bits). Two situations must be differentiated here:

1. Access using large real or absolute addresses.

   The 4 GB segment to be automatically considered for every subsequent read access with CURRENT.ATYPE=REAL or =ABS_ADDRESSING, i.e. for a real or absolute address interpretation, must be passed in CURRENT.SEGMENT.

2. PAM access on a large file.

   If the opened diagnosis object is a PAM file, absolute addresses must be specified for the localization of data. These addresses consist of the PAM page number P and the relative displacement D and are calculated using the following formula:

   $A = (P - 1)*2048 + D$ with $0 \leq D \leq 2047$

   In the case of large PAM files (a maximum of 534 773 760 pages are possible), more than 4 bytes may be needed for A. A PAM file is therefore subdivided into a total of 256 segments of 2 097 152 pages each.

   In order to find data, you will always need to specify the 4 GB segment (in CURRENT.SEGMENT) and the relative address within that segment in a PRODAMP procedure.

   To calculate the segment for a specified PAM page, you should use unsigned arithmetic (UNSIGNED_ON) as illustrated in the example given below.

   CURRENT.SEGMENT is maintained on a local procedure basis and is initially set to 0.

   In most cases (address < 4 GB), an explicit assignment is not needed. The provision of a negative value or a value greater than 255 always leads to a runtime error.

Example 1 for CURRENT.SEGMENT: Conversion of PAM page number to segment and address

```
UNSIGNED_ON;
   " The PAM page PAM_PAGE is to be converted to a 4 GB segment number
   - which is directly placed in CURRENT.SEGMENT -
   and a 32-bit address 'BYTE_ADDRESS'"

IF PAM_PAGE > X'1FE00000' THEN
   " Every BS2000 file has at most X'1FE00000' pages "

MESSAGE ('** ERROR: PAM PAGE TO LARGE **');

RETURN WINDOW=CURRENT.WNDNO;

END IF;

CURRENT.SEGMENT := (PAM_PAGE-1)/X'200000';
   "divided by 2 raised to (32-11)"

BYTE_ADDRESS    := (PAM_PAGE-1)*X'800';
   "multiplied by 2 raised to 11"
```

Example 2 for CURRENT.SEGMENT: Absolute read on the address X'1 8000 0000'

```
CURRENT.A_TYPE  := ABS_ADDRESSING;

   CURRENT.SEGMENT := 1;

   A:=X'80000000';
    "Address in the 44 GB segment"

   ARRANGE .FULLWORD: LENGTH=4,TYPE=NUMERIC,OFFSET=0;

Value := A.FULLWORD;
   "Access on a word at address X'000000'"

   IF CURRENT.ERROR = 0 THEN

    "etc."
```

CURRENT.SPID

CURRENT.SPID must be set to the required value for SPID (8-character string) when data spaces are accessed (see CURRENT.ATYPE).

CURRENT.TIME

CURRENT.TIME contains the CPU time, in milliseconds, that has elapsed since LOGON in the form of a numeric value with a length of 4. If the maximum time (approx. 25 days) is exceeded, the maximum value is returned and CURRENT.ERROR is set to a value other than 0.

CURRENT.TID, CURRENT.TSN and CURRENT.ITN

The pseudo-symbols CURRENT.TID and CURRENT.TSN permit access to the current task. CURRENT.ITN refers to the rightmost halfword of the TID, which unambiguously identifies the task at diagnosis time.

In the case of system, user and area dumps, the current task is always the dump task and cannot be changed. For SLED and SNAP dumps or in the active system, the current task is determined either by selecting a task in the status window (W2), by entering a TID (or ITN) or TSN in the header line of a window or, in PRODAMP, by calling the procedure NEW_TASK. The current task can be addressed via CURRENT.TID, CURRENT.ITN or CURRENT.TSN. All accesses to task-specific tables (TCB, JCB, etc.) and to addresses in the user address space always refer to the current task.

CURRENT.TID and CURRENT.ITN are numeric, while CURRENT.TSN is a 4-character string.

CURRENT.TSN

See CURRENT.TID.

CURRENT.VERSION

The pseudo-symbol CURRENT.VERSION contains the current BS2000 version of the diagnosis object in the form of a string with a length of 4. The format of the string is XX.X, for example 20.0. If a file is opened as an ordinary PAM file, the version in question is the version of the current operating system.

CURRENT.WNDNO

The pseudo-symbol CURRENT.WNDNO contains the number of the current diagnostic window, i.e. of the window which would appear at the top in a subsequent screen output. Since ARRANGE changes the order of the windows, CURRENT.WNDNO can be used, for example, to set the PRODAMP window as the current window again if an error occurs during window assignment.

The pseudo-base PARAMETER

Parameters can be transferred to PRODAMP procedures when calling from the DAMP program level or as a subprocedure of a PRODAMP procedure (see also section “Working with procedures (special window: PROC)” on page 299).
During the call, a parameter area is generated from the parameters, and contains a list of the parameter values. In the parameter area, numeric data values and bit patterns are always entered right-justified in a 4-byte field.
The first static call from a PRODAMP procedure is used by the compiler to define the

parameter area. The call consists of the name of the called procedure and a list of the current parameters enclosed in brackets.
PRODAMP also supports subprocedures with no parameters. Recursive calls are permitted.

In order to access the parameters, ARRANGE symbols must be defined with the PRODAMP statement. These symbols can then be addressed with the aid of the pseudo-base PARAMETER.

Example 1 for the pseudo-base PARAMETER

The GETOPC procedure is to return the operation code found at the address MOD + ADD in the symbol OPC.

Procedure GETOPC:

```
ARRANGE
.MOD : TYPE = STRING, RELATIVE = 0, LENGTH = 8;
.ADD : TYPE = NUMERIC,RELATIVE = 8, LENGTH = 4;
.OPC : TYPE = NUMERIC,RELATIVE =12, LENGTH = 4;
END ARRANGE ;
ARRANGE
.BYTE : TYPE=NUMERIC,RELATIVE=0,LENGTH=1;
END ARRANGE ;
A := ADDRESS ( PARAMETER.MOD,'CP' ) + PARAMETER.ADD ;
PARAMETER.OPC := A.BYTE ;
```

Procedure call:

```
MOD := 'DOPEN' ;
ADD := X'4ADC' ;
GETOPC ( MOD, ADD, OPC ) ;
IF OPC = X'47' THEN
.....
```

The structure variable PARAMETER.XX may be assigned a value. This value is assigned to the parameter variables used by the calling procedure at the same place.

At compilation time, the types and lengths of the parameter symbols and of all other structure variables are not known (see section “Working with procedures (special window: PROC)” on page 299).

Example 2 for the pseudo-base PARAMETER

The data is transferred with the default values “numeric” and “length 4”, even though other values were specified.:

```
ARRANGE
   .DST:TYPE=STRING,RELATIVE=0,LENGTH=8;
   .SRC:TYPE=STRING,RELATIVE=8,LENGTH=8;
END ARRANGE
PARAMETER.DST:=PARAMETER.SRC;
```

The correct assignment can be ensured by using an auxiliary variable:

```
STR:=' ' * 8;
STR:=PARAMETER.SRC;
PARAMETER.DST:=STR;
```

In the case of a call from the DAMP program level, parameters are passed by means of the RESUME-PRODAMP-PROGRAM or START-PRODAMP-PROGRAM statement. However, results cannot be returned to the calling level in this case; results can be returned only if a procedure calls another procedure. It is nevertheless possible to write procedures which work correctly, regardless of whether they are called from the DAMP program level or from another procedure, by interrogating CURRENT.LEVEL in order to identify the level at which the procedure is running.

This means that all parameters are passed by the “call by reference” method, i.e. the address of the parameter is passed to the procedure. However, literals and expressions can also be passed directly as parameters. The corresponding structure variables can also be described in the called procedure, but they cannot be used to return values to the calling procedure.

Various parameter lists can be defined by means of ARRANGE and subsequently overlaid on the pseudo-base PARAMETER after an INTERRUPT statement. Admittedly, in a later RESUME statement it is necessary to specify all the parameters which are evaluated after the associated INTERRUPT statement, but this method makes it possible to program a “guided dialog”.

Example 3 for the pseudo-base PARAMETER

```
ARRANGE
.TSN : TYPE=STRING,LENGTH=4, RELATIVE=0;
END ARRANGE;
MESSAGE ( 'Please enter TSN. (RESUME window,''tsn'')' ) ;
INTERRUPT;
TASK := PARAMETER.TSN ;
ARRANGE
.NUM : TYPE=NUMERIC,LENGTH=4,RELATIVE=0;
END ARRANGE;
MESSAGE ( 'Please enter number of passes. (RESUME window,num)');
INTERRUPT;
PASSES := PARAMETER.NUM ;
```

In a case of a symbolic access with the pseudo-base PARAMETER, it is only checked whether the datum to be accessed lies within the parameter area supplied by the caller, and if this is not the case, the program aborts with a corresponding runtime error message (see Examples 4 and 5).

Other errors - e.g. when the types of the call parameters do not match the fields defined with ARRANGE in the called procedure - cannot be detected by DAMP and can thus result in unpredictable side-effects at runtime.

You can query the overall length of the parameter area in the called procedure with the standard function LENGTH. This allows you to respond to missing parameters in the called procedure, for example.

Example 4 for the pseudo-base PARAMETER: Checking the length of the parameter area

The following PRODAMP program is to be called with two parameters, an address and an optional counter to be specified. If the second parameter is not supplied, a default value is to be used for the calculation instead.

```
"Layout of the parameter block (the 2nd. parameter may be missing )"
ARRANGE .ADDR_OF_STRUCT : OFFSET=0,LENGTH=4,TYPE=NUMERIC; "1st.parameter"
  .COUNTER : OFFSET=4,LENGTH=4,TYPE=NUMERIC; "2nd. parameter"
END ARRANGE;
" Transfer of parameters to the variables ADDR_OF_STRUCT and COUNTER."
" If counter is not specified, set COUNTER=100."
L:=LENGTH('*PARAMETER','DS');
IF CURRENT.ERROR <> 0 THEN
MESSAGE (' Program aborted. DAMP Version V4.2 or later required ');
RETURN WINDOW=CURRENT.WNDNO;
ELSIF L=4 THEN
ADDR_OF_STRUCT:= PARAMETER.ADDR_OF_STRUCT;
COUNTER       := 100;
ELSIF L=8 THEN
ADDR_OF_STRUCT:=PARAMETER.ADDR_OF_STRUCT;
COUNTER:=PARAMETER.COUNTER;
ELSE
MESSAGE ('Program aborted. Invalid parameter supplied');
END IF;
" etc. ( ADDR_OF_STRUCT and COUNTER are now supplied )"
```

Example 5 for the pseudo-base PARAMETER: Artificial parameter area

In some cases, it is useful to provide a special work area in the parameter area for a PRODAMP program, which can then be “freely” accessed. This can very easily be implemented by calling the program a second time, but now with an extended parameter area:

```
PROZEDUR XY

"Parameter layout"
ARRANGE :
.ADDR_IN: TYPE=NUMERIC,LENGTH=4,OFFSET=0;
.WORKAREA: TYPE=STRING,LENGTH=128,OFFSET=4;
END ARRANGE;
"Only ADDR_IN should be supplied for a direct call"

L:=LENGTH('*PARAMETER','DS');
WORKAREA := #'00'*64;
IF L = 4 THEN
   XY(PARAMETER.ADDR_IN,WORKAREA);  "Recursive call"
ELSIF L <> 4+64 THEN
   MESSAGE(' Invalid parameter supplied or DAMP prior to V4.2 ');
END IF;
"A work area initialized with binary zero is now available to the program
XY in the parameter area."
```

5.7.3.9 Predefined variables

PRODAMP provides two predefined variables, OPC_TABLE and SVC_TABLE, to control access to internal DAMP tables. On the one hand, these variables can be used as normal numeric variables, i.e. you can assign values to them and use them in arithmetic expressions. However, the length of the variables is less than that of normal numeric variables and thus not every numeric variable can be assigned.
An assigned value is, however, also used as an index for localizing an entry within a DAMP table. This index can then be used to address symbols which describe this entry.

OPC_TABLE

This variable is 2 bytes long and is used as an index for an entry in the DAMP table which describes the instruction code and is used during disassembly. If the contents of OPC_TABLE are less than 256 (i.e. if only one byte is used), they are understood to be the operation code for an instruction. If the contents are greater than 255 (if 2 bytes are used), they are understood to be the first two bytes of an instruction, the second byte being a subcode. An entry in the instruction table is described by the following DSECT:

```
INST     DSECT
INSTTYPE DS    X           INSTRUCTION TYPE
INSTNO   EQU   0              NO VALID INSTRUCTION
INSTRR   EQU   4              RR INSTRUCTION
INSTRX   EQU   8              RX INSTRUCTION
INSTRS   EQU   12             RS INSTRUCTION
INSTSI   EQU   16             SI INSTRUCTION
INSTSS   EQU   20             SS INSTRUCTION
INSTUN   EQU   24             UNKNOWN INSTRUCTION TYPE
INSTFLAG DS    X           FLAG
INSTPRIV EQU   X'80'          PRIVILEGED OPERATION
INSTSVAL EQU   X'40'          SUBFUNCTION VALID/AVAILABLE
INSTSVMN EQU   X'20'          SUBFUNCTION MNEMONIC VALID
INSTPSMN EQU   X'10'          PSEUDO MNEMONIC AVAILABLE
INSTFPI  EQU   X'08'          FLOATING POINT INSTRUCTION
INSTSPEC EQU   X'04'          SPECIAL OPERATION
INSTADW  EQU   X'03'          ACCESS DOUBLE WORD
INSTAWD  EQU   X'02'          ACCESS WORD
INSTAHW  EQU   X'01'          ACCESS HALFWORD
INSTXCPT DS    X           EXCEPTIONS
INSTOP1  EQU   X'80'          OPERAND 1 EXCEPTION
INSTOP1M EQU   X'40'          OPERAND 1 = MASK/R0=0 IF RR,RX
INSTOP1E EQU   X'20'          OPERAND 1 = EVEN/EXTENDED
INSTOP2  EQU   X'10'          OPERAND 2 EXCEPTION
INSTOP2M EQU   X'08'          OPERAND 2 = MASK/R0=0 IF RR
INSTOP2E EQU   X'04'          OPERAND 2 = EVEN/EXTENDED
INSTOP3  EQU   X'02'          OPERAND 3 EXCEPTION
INSTOP3M EQU   X'01'          OPERAND 3 = MASK
INSTOPC  DS    X           OPERATION CODE
INSTSVC  EQU   X'0A'          OPERATION CODE = SVC
INSTRS2  EQU   X'20'          RS INSTRUCTION WITH 2 SIZES
INSTOMN  DS    CL5         INSTRUCTION MNEMONIC
INSTO2M  DS    X           MASK FOR OPERAND 2
INSTFCT  DS    0XL6        SUBFUNCTION
INSTFCD  DS    X           FUNCTION DISPLACEMENT
INSTMSK  DS    X           FUNCTION MASK
INSTFCU  EQU   X'F0'          FUNCTION CODE IN UPPER HALFBYTE
INSTFCL  EQU   X'0F'          FUNCTION CODE IN LOWER HALFBYTE
INSTFCF  EQU   X'FF'          FUNCTION CODE IN FULL BYTE
INSTPTR  DS    A           FUNCTION POINTER
         ORG   INSTFCT
INSTFPT  DS    X           FUNCTION PSEUDO TYPE (RR ONLY)
INSTFTM  EQU   X'FC'          FUNCTION MASK FOR PSEUDO TYPE
INSTFTD  EQU   X'03'          FUNCTION MASK FOR DISPLACEMENT
INSTFMN  DS    CL4         FUNCTION MNEMONIC
INSTILEN EQU   *-INST      ITEM LENGTH
```

The following example is intended to illustrate how to address entries in the relevant DAMP table using the variable OPC_TABLE. As a prerequisite, a disassembly table must have been assigned, as is done when a diagnosis object is opened. If no such table has been assigned, you will need to declare one with the MODIFY-OBJECT-ASSUMPTIONS statement.

Example for OPC_TABLE

```
OPC_TABLE := X'B223';
MNEMO     :=  ' '*7;
PSEUDO    :=  'none';
INSTSVAL  := P'40';
INSTSVMN  := P'20';
INSTTYPE  := OPC_TABLE.INSTTYPE;
IF CURRENT.ERROR <> 0 THEN
   MESSAGE ( 'No instruction table available.' );
   RETURN;
END IF;
IF INSTTYPE = 0 THEN
   MNEMO := 'invalid';
ELSIF INSTTYPE = 24 THEN
   MNEMO := 'unknown';
ELSE
   MNEMO := OPC_TABLE.INSTOMN;
END IF;
IF INSTSVAL + INSTSVMN IN OPC_TABLE.INSTFLAG THEN
   IF OPC_TABLE > 255 THEN
      PSEUDO := OPC_TABLE.INSTFMN;
   ELSE
      PSEUDO := '    ';
   END IF;
END IF;
MESSAGE ( 'Mnemonic: '+MNEMO+' Pseudo: '+Pseudo );
```

SVC_TABLE

This single-byte variable is used as an index for an entry in the internal DAMP SVC table, which contains an 8-byte mnemonic for each SVC. The symbol file does not include a DSECT for these entries. The entries must be described using an ARRANGE statement as illustrated by the example below.

Example for SVC_TABLE

```
SVC_TABLE := X'5C';
ARRANGE .MNEMO : TYPE = STRING, LENGTH = 8, OFFSET = 0;
END ARRANGE;
MNEMO := ' '*8;
MNEMO := SVC_TABLE.MNEMO;
IF CURRENT.ERROR <> 0 THEN
   MESSAGE ( 'No SVC table available.' );
ELSE
   MESSAGE ( 'SVC ' + HEX_STRING(SVC_TABLE,2)+' = '+MNEMO );
END IF;
```

5.7.3.10 Standard procedures

Overview

The standard procedures incorporated in PRODAMP can only be called from within procedures. The syntax of the procedure calls corresponds to the Pascal syntax.

PRODAMP makes use of the following standard procedures:

Procedure name	Function
COMMAND	Issue DAMP statements from within a procedure
DMP_#REFRESH	Refresh the data area
DUMP_MEMORY	Output a memory area to SYSLST
ENTER_MODULE	Provide an interface between PRODAMP procedures and Assembler modules
EXTRACT	Manipulate strings
INSERT	Manipulate strings
LIST	Output a string to SYSLST
MESSAGE	Output a message
NEW_TASK	Set a new 'current' task
NEXT_WINDOW	Switch to a PRODAMP procedure in the next visible window of the DAMP screen
READ	Read from an EDT area
READ_WINDOW	Interrupt the PRODAMP procedure and allow entries or markings to be made in a diagnostic window
REFERENCE	Define a symbol as an element of a substructure
SET_HEADER	Create a header for a listing
UNSIGNED_ON	Turn on unsigned arithmetic
UNSIGNED_OFF	Turn off unsigned arithmetic
WRITE	Write to an EDT area

Table 13: Overview of PRODAMP standard procedures

The specified names should not be used for separate procedures, as this may lead to the program being interpreted incorrectly.

COMMAND
Issue DAMP statements

The standard procedure COMMAND enables the user to issue DAMP statements from within a PRODAMP procedure, e.g. in order to assign a private symbol file.

Procedure call

Operation	Operands
COMMAND	(text)

Operands

text — This specifies the text of the DAMP statement. “text” must be a string expression and must contain the statement in the form in which it was entered in the DAMP batch task or procedure.

“text” cannot be any of the following:
- REPEAT-SESSION
- RESUME-PRODAMP-PROGRAM
- SHOW-LAST-STATEMENT
- START-OPTION-DIALOG
- START-PATTERN-SEARCH
- START-PRODAMP-EDITOR
- START-PRODAMP-PROGRAM
- START-STATEMENT-SEQUENCE

DMP_#REFRESH
Refresh data areas

The standard procedure DMP_#REFRESH enables you to refresh data areas within a PRODAMP procedure. This may be required every now and then when diagnosing the active system.

Procedure call

Operation	Operands
DMP_#REFRESH	

DUMP_MEMORY
Output memory area

The standard procedure DUMP_MEMORY can be used to output a memory area to SYSLST in one of the standard dump formats. All parameters must be numeric expressions.

Procedure call

Operation	Operands
DUMP_MEMORY	(address,relad,length)

Operands

address — Specifies the start address of the area. Depending on the value of CURRENT.ATYPE, the virtual (default value), real, absolute or HSA memory or areas from data spaces are output.

In the case of large real and absolute addresses, only a value within a 4 GB segment can be specified in “address”. The associated segment must be specified in CURRENT.SEGMENT (see page 256). CURRENT.SEGMENT is maintained on a local procedure basis and is preset to “0”.

relad — Sets an initial value for the address relocation output in the list. If “relad” is set to 0, the addresses are output relative to the start of the area. (DAMP uses this format, for example, for output of the TCB.)
If a negative value is specified for “relad”, no relative addresses are output (used by DAMP, for example, for output of full pages).

length — Specifies the length of the area to be output.

Example

```
A := .ETCBTFT; DUMP_MEMORY (A, 0, LENGTH ('IDMTFT','DS'))
```

The first TFT is output to SYSLST with the length of DSECT “IDMTFT” (see also the description of the standard function LENGTH, page 294).

ENTER_MODULE
Call modules

The standard procedure ENTER_MODULE provides an interface between PRODAMP procedures and Assembler modules. It is also possible to branch to modules created with other languages provided that they conform to the conventions described below.

Procedure call

Operation	Operands
ENTER_MODULE	(module, par1, par2, ...)

Operands

module — Indicates the *module* to be called. The system expects *module* to be contained as an R type element in the PRODAMP library which has been defined as the object library. *module* must be a string type expression containing the name of the module in uppercase characters. Only the first 8 characters are evaluated. If *module* contains less than 8 characters, it is padded with blanks until it is 8 characters in length.

par1, par2, ... — This is the list of parameters to be made available to the called *module*. Each parameter must be an identifier for a PRODAMP variable or (in more general terms) a PRODAMP expression. These expressions or variables can be of any type. The list (par1, par2, ...) can be empty.

The registers are set as follows when the system branches to module:

R1 points to a parameter area supplied with values by PRODAMP.

R13 points to an 18-word (72-byte) area created by PRODAMP where the registers can be saved.

R14 contains the return address.

R15 contains the branch address.

When control is returned from the module, PRODAMP expects registers R1 to R12 to contain the same values they contained when control was passed to the module.

The parameter area has the following format:

Byte 0-1 Total length of the parameter area.

Byte 2-3 Contains the value 0.

Byte 4-11 Name of the module called.

Byte 12-n Transfer area containing the values of the parameters par1, par2,... in unbroken sequence.

- Numeric parameters and pattern type parameters are always 4 bytes in length.
- The length of a string type parameter depends on how it was defined in the PRODAMP procedure (1-133 bytes).

The entire parameter area must not exceed the length of a 4K page (4096 bytes). This means that the total length of the transfer area must not be greater than 4084 bytes.

When control is returned from the module, each variable which was passed as a parameter is updated using the value from the transfer area. This means that ENTER_MODULE also provides write access to PRODAMP variables.

ENTER_MODULE can also be used to start a module that was loaded in some other way. Thus, you can use the PRODAMP procedure COMMAND to issue the DAMP statement LOAD-MODULE in which a load library can be specified.

An entered module is unloaded on returning to the PRODAMP procedure only if it was not loaded with LOAD-MODULE. The loading of modules with the DAMP statement LOAD-MODULE (which is also possible from within PRODAMP via the COMMAND statement) can thus reduce the runtime of PRODAMP procedures considerably if these modules are called frequently. Furthermore, a load library can be specified in the LOAD-MODULE statement.

Examples

Example of a PRODAMP procedure that enters the ASSEMBLER module “TEST”, which changes the string “CARSICK” to “SEASICK”:

PRODAMP procedure

```
STR := 'CARSICK';
ENTER_MODULE ( 'TEST', STR );
MESSAGE ( STR );
```

Assembler module TEST

```
TEST     CSECT
TEST     AMODE ANY
         USING *,15
         STM   14,12,12(13)
         MVC   12(3,1),='SEA'
         LM    14,12,12(13)
         BR    14
         END
```

The example also shows how to save and restore registers. (In this case, this is in fact unnecessary, since TEST does not change the registers.)

Example of a PRODAMP procedure to reduce the runtime

```
COMMAND ('LOAD-MODULE *P-U-O-L(TEST)');
"The error case leads to the runtime error (Abort)"
 MESSAGE ('The module TEST was loaded');
"No unloading now occurs for ENTER_MODULE"
N := 0;
STR := ' '*4;
WHILE (N<100) DO
    ENTER_MODULE ('TEST',STR); N := N+1;
END WHILE;
```

EXTRACT
Manipulate strings

The standard procedure EXTRACT transfers as many characters from a source string, starting at a specified position, to a target string as will fit into this target string. If the length of the target string is greater than the number of characters to be transferred, the remaining characters in the target string are unaffected.

Procedure call

Operation	Operands
EXTRACT	(target,source,position)

Operands

target — Must be an identifier for an initialized variable of the type string. This variable contains the target string.

source — Specifies the source string (as a string type expression).

position — Specifies the position of the first character to be transferred within the source string. The first character of the source string is located at position 0.

Examples

```
A := 'XXXX' ;
EXTRACT ( A, 'Output for TSN 1234 under TSOS',15 );
```

Once the statement has been executed, A contains the text “1234”.

```
A := 'without problems';
EXTRACT ( A, 'This will probably not work.',19 );
```

Following execution of the EXTRACT procedure, A contains the string “not work.roblems”. Both examples assume that A has already been initialized by the statement shown.

INSERT
Manipulate strings

INSERT replaces the characters of a target string, starting at a specified position, with the characters of a source string. Characters are replaced until the last character from the source string has been transferred or the last character in the target string has been replaced.

Procedure call

Operation	Operands
INSERT	(source,target,position)

Operands

source — Specifies the source string (expression of the type string).

target — Must be an identifier for an initialized variable of the type string. This variable contains the target string.

position — Specifies the position of the character as of which the target string is to be overwritten.
“position” is specified relative to the start of the target string, i.e. the first character in the target string has the position 0.

Examples

```
A := 'Output for TSN XXXX under user ID $$$$$$$$.';
INSERT ('1234',A,15); INSERT ('TSOS    ',A,34);
```

Once the statements have been executed, A contains the following text:
“Output for TSN 1234 under user ID TSOS.”

```
A := 'This will probably not work.'
INSERT ( 'function.',A,22 );
```

Once the statements have been executed, variable A contains the text:
“This will probably not funct”

Both examples assume that A has already been initialized by the statement shown.

LIST
Output strings to SYSLST

The standard procedure LIST serves to output a string to SYSLST.

Procedure call

Operation	Operands
LIST	(string[,skiplines])

Operands

string — Specifies the string to be output.

skiplines — Defines how many empty lines are to be inserted before the string.
$0 \leq$ skiplines ≤ 15. If no value is specified, then skiplines = 0.

MESSAGE
Display message on screen

When issued without the optional numeric parameter “line”, the standard procedure MESSAGE causes a specified text to be displayed in one of the two message lines in the DAMP screen mask (lines 2 and 3) as soon as the PRODAMP procedure is terminated and the screen is refreshed. If the value 1 or 2 is assigned to the line parameter, the specified text is output to screen **immediately**.

line parameter not specified:

As soon as the buffer for the two message lines is full, any additional messages are ignored. For this reason, it is advisable to follow a MESSAGE call with either an INTERRUPT or a RETURN statement. If INTERRUPT is used, the user can read the text and then, if desired, resume the procedure with RESUME. However, if the message is an error message and the procedure is to be aborted, it is better to use RETURN.

line parameter specified:

This allows interim messages providing information on the current state of processing in the procedure to be issued during PRODAMP procedures which run over longer periods.

Procedure call

Operation	Operands
MESSAGE	(text[,line])

Operands

text — Text to be displayed.

line — stands for a numeric expression with the value 0, 1 or 2. The values indicate the message lines to which the messages are to be written.
Message line 1 ≙ line 2 of the screen
Message line 2 ≙ line 3 of the screen
A specification of 0 for line corresponds to the procedure call MESSAGE (text).

NEW_TASK
Set current task

The standard procedure NEW_TASK sets a new current task as defined by DAMP. This is meaningful only for SLED dumps, SNAP dumps and the active system. All accesses to task-specific tables (ETCB, EJCB etc.) or to addresses in user memory then refer to the new task.

Procedure call

Operation	Operands
NEW_TASK	(task[,map])

Operands

task — Specifies the new current task.

"task" must be a string expression or a numeric expression. NEW_TASK works differently in both cases.

If task is a *string expression*, it must contain a TSN (up to 4 characters in length). If a task exists with this TSN, it is set as the current task. If this is not the case, CURRENT.ERROR is set.

If task is a *numeric expression*, the last 3 half-bytes of this value are interpreted as an ITN. If an active task with this ITN exists in the diagnosis object, this task is set. If this is not the case, the subsequent task (in the TLT) is set. CURRENT.ERROR is only set if it is not possible to find a task in this
manner.
If the numeric expression is an identifier for a numeric variable, it returns the TID set. This allows you to work through all active tasks sequentially (see example 3, "Changing the current task", on page 310).

map — Specifies whether the existing system overview (CSECT map) is to be extended to include the overview of the nonprivileged subsystems for the new task.
The value TRUE or FALSE can be used for the parameter map. The value TRUE requests the additional CSECT map. No specification is equivalent to the specification FALSE.

The specification TRUE results in decreased performance and can easily cause a memory overflow on systems with a large number of active tasks. The optional parameter map should therefore only be used when absolutely necessary.

Example

```
NEW_TASK ( '0A33' ); ————————————————————————————————————————————— (1)
TASK_ID := X'AB'; ———————————————————————————————————————————————— (2)
NEW_TASK ( TASK_ID );
```

(1) The task with the TSN 0A33 is selected first. If this task does not exist, CURRENT.ERROR is set.

(2) Then the task with the ITN X'AB' is selected as the current task. If this task does not exist either, the next task in the TLT (which could be X'AE') is selected and its TID is returned in TASK_ID. CURRENT.ERROR is set only if task X'AB' is not active and there are no more active tasks in the TLT.

NEXT_WINDOW
Switch to next visible window

NEXT_WINDOW can only be called if READ_WINDOW has previously been called successfully. It provides the pseudo symbols INFIELDS.xxx for the next diagnostic window in the screen read by READ_WINDOW. Please refer to READ_WINDOW for further information.
If the DAMP screen does not have any more visible windows, the pseudo symbol CURRENT.ERROR is set to a value other than 0.

The pseudo symbol INFIELDS.COMMAND can always be used to access the command line of the DAMP screen, irrespective of the viewed window.

Procedure call

Operation	Operand
NEXT_WINDOW	

Example

```
READ_WINDOW; ————————————————————————————————————————————————— (1)
WHILE (CURRENT.ERROR = 0) DO ————————————————————————————————— (2)
   WRITE (DEC_STRING (INFIELDS.WNDNO)); ———————————————————————— (3)
   NEXT_WINDOW; ————————————————————————————————————————————— (4)
END WHILE;
```

(1) The procedure is interrupted. The procedure is reactivated when the P13 key is pressed, and the entries made in the last screen are stored internally. The topmost diagnostic window on the screen is the current diagnostic window to which INFIELDS.xxx refers.

(2) Aborts the procedure if NEXT_WINDOW does not display any further visible window.

(3) This allows the pseudo symbols INFIELDS.xxx to be evaluated for the current window. In the example, the window number is written to the EDT area.
Since INFIELDS.WNDNO always exists, it is not necessary to evaluate CURRENT.ERROR.

(4) Switch to next window.

READ
Read from EDT area

The standard procedure READ reads sequentially from the current EDT area and assigns the record read to a string variable text. If this string variable has not yet been initialized, a string with the maximum permissible length (133 bytes) is created. If the EDT line is too short, the variable is filled with blanks; if it is too long, surplus characters are ignored.

In addition to being used to access diagnostic data that is not in the diagnosis object (such as the REP file), this function can be used, for example, to store table layouts in separate files and to read them into the PRODAMP procedure, thereby doing away with the need for resource-intensive initialization operations.

The user is advised not to alternate between the procedures READ and WRITE, since it is possible that WRITE will change the current line number set internally by EDT. A subsequent READ operation could possibly return the wrong line.

Procedure call

Operation	Operand
READ	(text)

Operands

text — String variable to be assigned to the text read.
Maximum length: 133 characters.

READ_WINDOW
Interrupt PRODAMP procedure and allow entries or markings to be made in diagnostic window

The standard procedure READ_WINDOW interrupts the PRODAMP procedure. The PRODAMP procedure is reactivated after you press the P13 key. Until you do this, you can carry out any work required in the DAMP screens.
When the procedure is reactivated using the P13 key, the entries in the last DAMP screen are not passed to the DAMP screen, but to PRODAMP instead. The pseudo symbols INFIELDS.xxx (see below) make available entries in the topmost diagnostic window on the screen as well as entries in the command line. The standard procedure NEXT_WINDOW allows access to the entries in any subsequent diagnostic window (on the same screen). Calling the NEXT_WINDOW procedure a number of times in succession makes the entries from all the diagnostic windows on the screen (from top to bottom) available.

As well as entries, READ_WINDOW allows you to determine a number of values assigned to the windows (such as the window number). Not all entries are available. This applies particularly to entries in most of the special windows. Please refer to the list below for details. This list contains the permitted INFIELDS.xxx pseudo symbols.

Users can program their own interfaces for DAMP by incorporating READ_WINDOW and NEXT_WINDOW procedures in a PRODAMP procedure and always using the P13 key for transferring data. This allows implementation of new, user-specific DAMP statements.

Procedure call

Operation	Operands
READ_WINDOW	

The following pseudo-symbols can be accessed:

INFIELDS.ADDRESS
Contains any entry made in the **Absolute address** field in the header line of the viewed window; 4-digit numeric value.

INFIELDS.ASEL
Contains any entry made in the **ASEL** (Address Space Selector) field in the header line of the viewed window; 3-character string.

INFIELDS.ASID
Contains any entry made in the **ASID** (Address Space Identifier) field in the header line of the viewed window; 17-character string.

INFIELDS.COMMAND
Contains any entry made in the DAMP statement line; 72-character string.

INFIELDS.LAYOUT
Contains any entry made in the **Window layout** field in the header line of the viewed window; 3-character string.

INFIELDS.LENGTH
Contains any entry made in the **Length** field in the header line of the viewed window; 1-digit numeric value.

INFIELDS.MARK1 to **INFIELDS.MARK6**
Contains any addresses marked in the viewed window; 4-digit numeric value (for each address).

INFIELDS.RELATIVE
Contains any entry made in the **Relative address** field in the header line of the viewed window; 4-digit numeric value.

INFIELDS.STACK
Contains any entry made in the **Stack number** field in the header line of the viewed stack window; 4-digit numeric value.

INFIELDS.SYMBOL
Contains any entry made in the **Symbol** field in the header line of the viewed window; 31-character string.
(For reasons of compatibility, the last (i.e. 32nd) character of the **Symbol** field is ignored.

INFIELDS.TID
Contains any entry made in the **TID** field in the title line of the viewed dump window; 4-digit numeric value.

INFIELDS.TSN
Contains any entry made in the **TSN** field in the title line of the viewed dump window; 4-digit numeric value.

INFIELDS.WNDNO
Contains the window number of the viewed window; single-digit numeric value.

INFIELDS.WNDTSK
Contains the TID to which the data in the viewed window belongs; 4-digit numeric value.

Notes

– If the pseudo-symbol is queried even though no entry has been made in the corresponding field in the DAMP window, the pseudo-symbol CURRENT.ERROR is set to a value other than 0. The pseudo-symbols WNDNO and WNDTSK are set implicitly. WNDNO is always valid and WNDTSK is valid whenever the window contains data from a user task. If the data is system data, the value contained in WNDTSK is not valid (CURRENT.ERROR is set).

– If the PRODAMP procedure interrupted with READ_WINDOW is resumed using the RESUME-PRODAMP-PROGRAM statement, the procedure is always aborted if an attempt is made to access one of the pseudo-symbols. In this event, an appropriate message is issued.

– The data stored in the pseudo-symbols is not destroyed if the PRODAMP procedure is interrupted normally using the INTERRUPT statement. It remains available after the procedure has been resumed (using the RESUME-PRODAMP-PROGRAM) statement. Each time a PRODAMP procedure is restarted, however, the data area accessed by the pseudo-base INFIELDS is set to an invalid value. This is also the case if a different procedure was started between the INTERRUPT and RESUME-PRODAMP-PROGRAM statements.

– Irrespective of the number of windows displayed on the screen, a maximum of 6 markers are transferred and assigned to the relevant windows with the READ_WINDOW and NEXT_WINDOW procedures. The markers are available in the PRODAMP procedure via the pseudo-symbols INFIELDS.MARK1 through INFIELDS.MARK6.

Example

The following procedure waits in the background until an address field is marked and transferred using the P13 key. If this happens, the procedure outputs the first word located at the marked address in the message line.

```
ARRANGE
 .WORD : OFFSET = 0, LENGTH = 4, TYPE = NUMERIC;
END ARRANGE;
B := 0;
WHILE B=B DO
  CURRENT.ERROR := 0;
  READ_WINDOW;
  WHILE CURRENT.ERROR = 0 DO
    A := INFIELDS.MARK1;
    MESSAGE ( HEX_STRING(A,8)+': '+HEX_STRING(A.WORD,8) );
    NEXT_WINDOW;
  END WHILE;
END WHILE;
```

REFERENCE
Localize symbol which is element of substructure

The standard procedure REFERENCE is only used in conjunction with the standard functions ADDRESS and LENGTH. The procedure is used for specifying references if the symbol to be processed using ADDRESS or LENGTH is an element of a substructure. The REFERENCE procedure must be called for each symbol which lies on the “path” to the required element. This must be done in the correct sequence (see example). Symbols must be specified as a string of up to 32 characters. These are collected by the REFERENCE calls, but not checked for validity. The strings are only checked when the ADDRESS or LENGTH function is called.

All REFERENCE calls are only valid locally within a procedure. If the calls are not resolved by calling ADDRESS or LENGTH when the procedure is terminated, the procedure is aborted. If the call is resolved, the referenced path is deleted. If the element is required again, you must make new REFERENCE calls.

Procedure call

Operation	Operand
REFERENCE	(symbol)

Operands

symbol Specifies the symbol to be localized; 31-character string.

Example

```
NKLCB_MDL.COPY_PARAMETER.USER_ADMINISTRATION.WAIT_FACTOR ———————————  (1)
REFERENCE ( 'NKLCB_MDL' );
REFERENCE ( 'COPY_PARAMETER' );
REFERENCE ( 'USER_ADMINISTRATION' ); ———————————————————————————————  (2)
A := ADDRESS ( 'WAIT_FACTOR', 'RF' ); ——————————————————————————————  (3)
```

(1) This SPL structure field is to be evaluated.

(2) The REFERENCE procedure is used three times to indicate the path (“NKLCB_MDL.COPY_PARAMETER.USER_ADMINISTRATION”) leading to the WAIT_FACTOR symbol.

(3) This returns the address of the WAIT_FACTOR symbol. The address is calculated relative to the first symbol of the chain of references (NKLCB_MDL in the example). Specifying “RF” indicates that REFERENCE calls were required to find this symbol.

SET_HEADER
Generate header for list output

SET_HEADER can be used to generate a header for a list output. This header is output for the first time when SET_HEADER is called; after this, it is output after each subsequent new page in the list until the text is changed by a new SET_HEADER call.

Procedure call

Operation	Operands
SET_HEADER	(string,skiplines,reservelines)

Operands

string — must be an expression of the type “string” and contains the text for the header.

skiplines — must be a numeric expression and specifies the number of lines to be skipped before the header is printed for the first time. $0 \leq$ skiplines ≤ 255.

reservelines — must be a numeric expression and has the following effect: if there are less than “reservelines” lines left on the current page before the next page break, a form feed to a new page is executed before the header is printed for the first time.
$0 \leq$ reservelines ≤ 255.
Specifying “255” forces a form feed.

Example

```
SET_HEADER ('TEXT',0,255); ————————————————————————————————————  (1)
SET_HEADER ('TFT FOR TASK'+CURRENT.TSN, 2, 20); ———————————————————  (2)
```

(1) This forces a form feed before the header.

(2) 2 lines are skipped before the header (one line is always skipped after the header). If there are less than 20 lines left on the current page, a form feed is executed.

UNSIGNED_ON and UNSIGNED_OFF
Enable and disable unsigned arithmetic

When performing calculations with addresses, it can sometimes be a disadvantage to work with signed arithmetic. For this reason, it is possible to select between unsigned and signed arithmetic for the numeric data type of PRODAMP by calling a standard procedure.

In the signed interpretation of 32-bit data, the leading bit has a value of -2^{31}; in the unsigned interpretation, its value is 2^{31}.

A PRODAMP main routine initially runs with signed arithmetic enabled (UNSIGNED_OFF, the default setting for PRODAMP). Unsigned arithmetic must be enabled with the procedure UNSIGNED_ON and can be disabled again with UNSIGNED_OFF.

On calling a subroutine, this arithmetic execution mode (i.e. signed or unsigned) is inherited by the subroutine. Changing the mode in the called routine has no effect on the arithmetic execution mode in the calling routine.

The arithmetic execution mode only affects calculations with the arithmetic data type of length 4 (32-bit data). The addressing of data objects is not affected.

Signed arithmetic (UNSIGNED_OFF)

Calculations with signed arithmetic are performed as usual in PRODAMP. If errors occur (overflow on addition, subtraction and multiplication; division by zero), the PRODAMP run is aborted with a runtime error.

Unsigned arithmetic (UNSIGNED_ON)

Overflows on addition, subtraction and multiplication are ignored (the result is "modulo 2^{32}", which is correct) and, in particular, **do not** result in a runtime error.
In the case of a division by zero, the PRODAMP run is aborted with a runtime error.
Comparisons in unsigned arithmetic are performed as binary "logical comparisons".

In the standard functions "DEC_BINARY - Convert decimal number" and "DEC_STRING - Convert numeric values", the switch to signed arithmetic with UNSIGNED_ON is ignored.

Procedure call

Operation	Operand
UNSIGNED_ON	
UNSIGNED_OFF	

WRITE
Write to EDT area

The standard procedure WRITE causes a text to be written to an EDT area (namely area 8, unless otherwise specified). This makes it possible, for example, to generate a table of system values in the EDT area and then evaluate this table in EDT.

The EDT output area can be modified at any time with the aid of WRITE (“@PROC nn”).

Strings which begin with the character “@” are interpreted as EDT statements. “@WRITE "filename"” can thus be used to save the contents of the EDT area to a file.
However, only EDT statements accepted in EDT F mode are accepted in the WRITE procedure. Other statements can only be issued by way of an EDT procedure file.

PRODAMP simulates the EDT statements **@PROC** and **@END**, albeit with the following restrictions:
– Only procedure files 1 to 9 can be used.
– If the system recognizes the number of a valid procedure file, no further syntax checking is carried out. Any further command entry is ignored.
– The @END command must not be abbreviated.
– It is not possible to chain @PROC or @END together with other EDT commands.

If old PRODAMP procedures which still use L mode EDT commands are recompiled under DAMP V4.7, they will no longer run.

The standard procedure WRITE may modify the current line number set internally by EDT. If it is followed by a READ procedure, the resultant line contents will be other than expected. For this reason, alternate use of WRITE and READ should be avoided.

Procedure call

Operation	Operands
WRITE	(text)

Operands

text Specifies the string expression to be written to the EDT area.

5.7.3.11 Standard functions

The standard functions contained in PRODAMP can only be called within procedures. The syntax of the function calls corresponds to Pascal syntax.

Function name	Function
ADDRESS	Return the address of a module or control block
DEC_BINARY	Convert a decimal string to a numeric variable
DEC_STRING	Decimal editing for printing
HEX_BINARY	Convert a hexadecimal string to a numeric variable
HEX_STRING	Hexadecimal editing for printing
LENGTH	Output the length of a module, control block or parameter area
LOCATION	Return the module name for an address
PATTERN	Convert a numeric value to a PATTERN type value
PCK_BINARY	Unpack packed numbers from a diagnosis object

Table 14: Overview of PRODAMP standard functions

The listed names should not be used for user-written procedures, since this may lead to misinterpretation of the program.

ADDRESS
Return address of module or control block

The standard function ADDRESS supplies the address of a module. Consequently, the result variable must be numeric. If the result variable has not yet been initialized, it is assigned the type “numeric” and the length “4”. ADDRESS can also be used to determine relative addresses of fields in a control block.

User program modules can normally only be found in user dumps.

Function call

Operation	Operands
ADDRESS	(modname, susyname)

Operands

modname — Specifies the name of the required module. 'modname' may not contain more than 32 characters; however, leading and trailing blanks are permitted before and after the module name.

In string types, a distinction is made between uppercase and lowercase notation. *modname* must therefore be specified in uppercase letters.

susyname — Specifies the subsystem in which the relevant module is to be localized. *susyname* is a string containing the name of a subsystem or one of the following symbolic names:

CP	Control Program, i.e. all class 1 and class 2 modules are searched for “modname”.
*PRIV	All privileged subsystems are searched for “modname”.
*NONPRIV	All nonprivileged subsystems are searched for “modname”.
*USER	Only the user CSECTs are searched for “modname”.
*ALL	The entire system is searched for “modname”.

If the relative address of a control block is to be determined, *modname* must contain the field name and *susyname* must contain the character string “DS”. If the name of a symbol is specified for *modname*, and if this symbol is the element of a substructure localized by REFERENCE calls, the character string “RF” must be entered for *susyname*.
(For further details see the example for the standard procedure REFERENCE on page 284.)

Example

```
A := ADDRESS ('DOPEN', 'CP') ;
MODULE := 'FAUTEM' ;
A := ADDRESS (MODULE, 'ARCHIVE');
A := ADDRESS ('EXVTDSSM','DS');
```

If the module or symbol is not found, the pseudo-symbol CURRENT.ERROR is given a value ≠ 0. In this case, the returned result is undefined.

DEC_BINARY
Convert decimal number

The standard function DEC_BINARY interprets the contents of a string as a decimal number and returns this number as data of the 4-byte numeric type. Any blanks before and after the relevant characters are ignored.

If the string does not contain a valid decimal value or if the value lies outside the limits for numeric variables, CURRENT.ERROR is set. In this case, the result of DEC_BINARY is undefined.

The enabling of unsigned arithmetic with the standard procedure UNSIGNED_ON is ignored.

Function call

Operation	Operand
DEC_BINARY	(string);

Operands

string String variable that is to be converted.

DEC_STRING
Convert numeric values

The standard function DEC_STRING converts a specified number into a decimal string. The enabling of unsigned arithmetic with the standard procedure UNSIGNED_ON is ignored.

Function call

Operation	Operands
DEC_STRING	(number[,length])

Operands

number — Specifies the number that is to be converted.

length — Specifies the length of the string generated. The decimal number is entered in the string right-justified and is padded to the left with blanks. If the specified length is insufficient, leading characters are truncated.

If you do not specify a length, only the significant characters are returned (compact format without leading blanks).

Exception

X := DEC_STRING(I) has the same effect as X := DEC_STRING(I,L), if X is a string variable which has already been initialized with the length L.

If you use DEC_STRING without specifying a length for initializing a variable or as a parameter for an expression to be passed to one of your own PRODAMP subroutines, 10 bytes are reserved for the result. In both cases it is better to specify a length explicitly.

Example

```
Task:                                          Result:
X := 'XXXXXXXX'; ──────────────────────────────────────────────────────  (1)
X := DEC_STRING (123);                         '     123'
X := 'XXX';
X := DEC_STRING (123456);                      '456' ──────────────────  (2)
X := 'ABC' + DEC_STRING (12) + 'XYZ';          'ABC12XYZ        ' ──────  (3)
X := 'ABC' + DEC_STRING (12,5) + 'XYZ';        'ABC   12XYZ' ──────────  (4)
```

(1) X initialized as an 8-byte string.

(2) X initialized as a 3-byte string.

(3) X not initialized. A string of the length 3 + 10 + 3 is produced.

(4) X not initialized. The “length” operand, however, is passed a value of 3 + 5 + 3 when the DEC_STRING function is called.

HEX_BINARY
Convert hexadecimal number

The standard function HEX_BINARY interprets the specified string as a hexadecimal number and returns the result as data of the 4-byte numeric type. Any blanks before and after the relevant characters are ignored.

If the string does not contain a valid hexadecimal value or if the value lies outside the limits for numeric variables, CURRENT.ERROR is set. In this case, the result of HEX_BINARY is undefined.

Function call

Operation	Operands
HEX_BINARY	(string);

Operands

string String variable that is to be converted.

HEX_STRING
Convert numeric values

The standard function HEX_STRING converts a specified number into a hexadecimal string.

Function call

Operation	Operands
HEX_STRING	(number[,length])

Operands

number — Specifies the number that is to be converted.

length — Specifies the length of the string generated. The hexadecimal number is entered in the string right-justified and is padded to the left with zeros. If the specified length is insufficient, leading characters are truncated.

If you do not specify a length, only the significant characters are returned (compact format without leading blanks). In the same way as with DEC_STRING, there are exceptions where no length is specified. Where necessary, the compiler reserves 8 characters for the result of HEX_STRING.

Examples

```
Task:                                          Result:

X := 'XXXXXXXX';
X := HEX_STRING (123)                         '0000007B'

X := 'XXX';  "too short"
X := HEX_STRING (4097)                        '001'

X not yet initialized
X := HEX_STRING (123,8)                       '0000007B'

X not yet initialized
X := HEX_STRING (123)                         '0000007B'
```

LENGTH
Output length of a module, control block or parameter area

The standard function LENGTH can be called in three variants:

– Variant 1: outputs the length of a module
– Variant 2: outputs the length of a control block (DSECT/SPL structure, substructure and field)
– Variant 3: outputs the length of the parameter area with which the current procedure was called

The result is of type “numeric” and has the length “4”.

Function call

Operation	Operands
LENGTH	(modname, susyname);

Operands

modname

Variant 1: 'modname' is the name of a module, whose length is to be determined. 'modname' can have a maximum of 32 characters, but blanks before or after the module name are allowed.

Variant 2: 'modname' is the name of a control block, whose length is to be determined; maximum 32 characters, but also with additional blanks allowed before or after the name.

Variant 3: 'modname' is the string '*PARAMETER'

susyname

Variant 1: The name of the subsystem in which the module is to be searched.

The following values of the string “susyname” have a special meaning:

CP	The control program is searched, i.e. among all CI1 and CI2 modules.
*PRIV	All privileged subsystems are searched.
*NONPRIV	All nonprivileged subsystems are searched.
*USER	Only the user CSECTs are searched.
*ALL	Everything is searched.

susyname	Variant 2:	'susyname' contains 'DS' or 'RF'. This indicates that the length of the control block or field is to be determined. 'RF' is used in cases where the field is an element of a substructure and used for localizing REFERENCE calls.
	Variant 3:	'susyname' contains 'DS', and “modname” is the string “*PARAMETER”.

Notes

1. The call format LENGTH('*PARAMETER','DS') sets the value of CURRENT.ERROR to 0 and returns the entire length of the parameter area with which the current procedure was called. This length is 0 if the call occurred without parameters.

   Examples for the use of LENGTH('*PARAMETER','DS') can be found starting on page 261.

2. When LENGTH is called for a module or control block, a subsequent examination of CURRENT.ERROR should indicate whether or not this module or control block was actually found.
   This allows you to check for the existence of a control block field in the currently loaded symbol elements before a symbolic access with that control block field and to respond accordingly.

Examples

```
L:=LENGTH('NCTVXVT','CP');
L:=LENGTH('EXVT','DS');
L:=LENGTH('EXVTPRD','DS');
L:=LENGTH('*PARAMETER','DS');


L:=LENGTH('MYDSECT','DS');
IF CURRENT.ERROR <> 0 THEN
" Error case; continue by dynamically loading required symbols via "
" COMMAND ('ADD-SYMBOLS ...'), for example, or error exit          "
END IF;
```

LOCATION
Return module name for address

The standard function LOCATION returns the name of the module in which the specified module name is located. Consequently, the result must be assigned to a string variable.

The module name is transferred in 32 characters at the most. If the string variable is too short to accommodate the module name, it is truncated. If it is longer than the module name, the remaining space is filled with blanks. If the target variable has not yet been initialized, it is assigned the type “string” and the length “8”.

If the specified expression does not match any address from a system module, LOCATION sets the pseudo-symbol CURRENT.ERROR. If the address is found in user memory, the target variable receives the value “ABSOLUTE”; if not, it receives the value “?”.

The displacement of this module from the start of the module can then be determined with the aid of the standard function ADDRESS.

Function call

Operation	Operands
LOCATION	(address, susyname);

Operands

address — Address of the module that is to be localized.

susyname — Subsystem in which the module is to be localized. susyname is a string that contains either the name of the subsystem or one of the following symbolic identifiers:

CP	The control program is searched, i.e. among all CI1 and CI2 modules.
*PRIV	All privileged subsystems are searched.
*NONPRIV	All nonprivileged subsystems are searched.
*USER	Only the user CSECTs are searched.
*ALL	Everywhere is searched.

Example

```
NAM := LOCATION ( ADDR, 'CP' );
REL := ADDR - ADDRESS ( NAM, 'CP' ) ;
```

PATTERN
Convert numeric value

The PATTERN function converts a numeric value to a value of the type PATTERN. If the result variable does not exist, it is assigned a length of 4 bytes.

A conversion of this type is required if, for instance, a loop is to be used to check data to see whether certain bits are set or not. Since no operations are available which, for instance, produce the value P'02' from P'01', this can only be done by converting numeric values.

Function call

Operation	Operands
PATTERN	(num)

Operands

num Specifies the numeric value to be converted.

Example

```
L := 1;
U := 1;
WHILE L < 8 DO
   IF PATTERN ( U ) IN .EXVTULM THEN
      .......
      .......
   END IF;
   U := U * 2;
   L := L + 1;
END WHILE;
```

PCK_BINARY
Unpack packed numbers

PCK_BINARY converts packed numbers into unpacked numbers. The result is numeric and has the length “4”.

If the string does not contain a packed number, CURRENT.ERROR is set.

Function call

Operation	Operand
PCK_BINARY	(string)

Operands

string Specifies the number that is to be unpacked.

5.7.4 Working with procedures (special window: PROC)

Every PRODAMP procedure can be called either from the DAMP program level or as a subroutine. In the first case, the procedure is called either after compilation by specifying MODE=Xqt or MODE=Go, by means of the DAMP statement RESUME-PRODAMP-PROGRAM (with the option of passing parameters), or using the DAMP statement START-PRODAMP-PROGRAM, should the PRODAMP procedure be in a library in the form of an object (here also with the option of passing parameters).

A PRODAMP procedure can be called as a subroutine only if it is stored in object form in a library or if it has already been compiled and is currently in a PROC window. The procedure is then called by specifying its name together with the list of parameters, if appropriate. The procedure name is identical to the element name of the PRODAMP object in the library (see section “Archiving private procedures” on page 307).

Creating private procedures

The creation of PRODAMP procedures comprises two components: editing and compilation.

In order to create a procedure, the PRODAMP compiler must first be assigned to one of the dump windows. This is done using the following statement:

```
START-PRODAMP-EDITOR WINDOW=<w>, SOURCE=filename
```

WINDOW Specifies the window (4 to 9 or 21 to 99) in which the PRODAMP procedure is to be edited, compiled and possibly executed.

SOURCE Specifies a file that already contains a PRODAMP source file. This file is then read immediately into the specified window. This specification is optional.

A window being used by a PRODAMP procedure is not reset when the dump file is switched by means of the DAMP statement OPEN-DIAGNOSIS-OBJECT.

Input fields in the PROC window

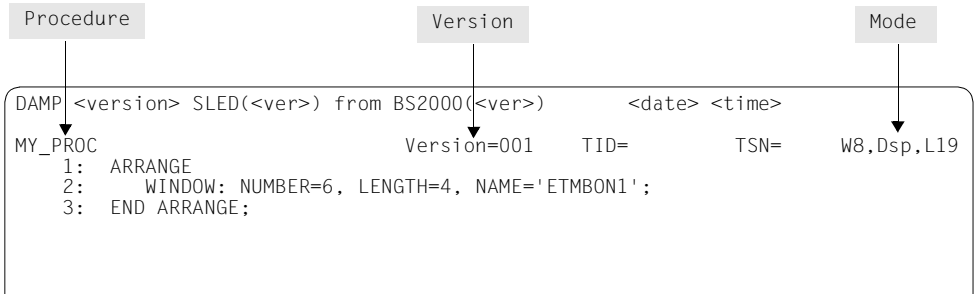


Figure 67: Dump window to which the PRODAMP compiler has been assigned

Within a procedure that has been read in or written, it is possible to scroll back (+nn) and forth (-nn) or position to a specific line (#nn) in the same way as in EDT. You can also enter text in the header line and in the text lines. Entries in the header line (“Procedure”, “Version” and “Mode”) initiate PRODAMP functions, while entries in the program lines are regarded as changes to these lines.

Meanings of the three fields in the header line

Mode Mode; specifies what is output in the dump window or what action the PRODAMP compiler is currently performing or is to perform. Entering one of the following strings and sending the screen off by hitting the DUE key triggers the corresponding action.

=Beg (Begin); positions the window to line 1 of the source file.

=Cmp (Compile); starts the compiler for the current contents of the window.

=Dsp (Display); default setting for modifying source file lines within the PRODAMP window.

=Edt (Edit); transfers the current PRODAMP procedure to an EDT area. Major changes to the PRODAMP source file should always be made in EDT. The PRODAMP window permits only simple editing functions (scrolling, overwriting, deleting). When the editor is terminated by means of END, HALT or RET, the edited source file is transferred back to the PRODAMP window and the EDT area is then cleared.

=Go is equivalent to “compile + execute + begin”.

=Inf (Inform); outputs a list of the source elements contained in the current source library and releases a lock set using Lck (Lock). If the keyword OBJECT is entered in the “Procedure” field, the display switches to the directory of the object elements. By the same token, entering SOURCE restores the original display.

=Lck (Lock); has the same effect as Read, but also locks the element of the source library to be read against concurrent accesses. The lock is released using Update, New or Inform. New and Inform do not change the original contents of the library. Only Update overwrites the old data.

=New clears the PRODAMP window so that a new PRODAMP procedure can be displayed.
Any lock set by Lck (Lock) is canceled. The system does not request confirmation.

The PRODAMP window is cleared without a request for confirmation.

=Rea (Read); an element is read from the selected source library. As for M=Wrt, the element name and the version are taken from the fields “Procedure” and “Version” in the header line. If no version is specified, DAMP automatically accesses the highest existing version.

=Sav (Save); stores a PRODAMP object in the selected object library. The name specified in the “Procedure” field is used as the element name, and the number specified in the “Version” field is used as the version number. An element with the same name and version will be overwritten without first asking for confirmation.

=Upd (Update); in the selected source library the element with the name specified in the “Procedure” field and the version specified in the “Version” field is overwritten. Any lock set using Lck is released. If there was no element with this version, an error message is issued. In this case, Mode=Wrt must be used.

=Wrt (Write); the current source is written to the library set via ASSIGN or to the default library. The name specified in the “Procedure” field is used as the element name, and the number specified in the “Version” field is used as the version number. If no version is specified, DAMP automatically assumes 001. An existing element with the same name and version will not be overwritten unless explicitly requested by way of Mode=Upd.

=Xqt (execute); executes a compiled PRODAMP procedure.

Procedure Displays the name of the procedure currently being output.

Version Specifies the version of the procedure with the specified name.

Useful information for practical application

The following sequence of actions is recommended when trying out PRODAMP for the first time:

– Start PRODAMP using START-PRODAMP-EDITOR.
– Switch to EDT via Mode=Edt.
– Edit the PRODAMP procedure in EDT.
– If desired, save the procedure by means of WRITE.
– Return to DAMP by entering HALT.
– Compile the procedure via Mode=Cmp.
– If necessary, correct any syntax errors in the PROC window or using EDT again.
– Execute the syntactically correct procedure via Mode=Xqt.
– If necessary, debug the procedure.

Common errors

– A symbol which can be localized automatically is entered without the period that forms part of the symbol. As a result, PRODAMP does not recognize this as a symbol, but interprets it as a variable.

– When using symbols, the programmer forgets that the types and lengths of the symbols are not known until runtime. If a value is assigned to a variable which has not been initialized, the variable can thus receive only the default type (numeric, length 4). It is therefore best to declare all variables by initializing them.

– After accessing the object to be diagnosed, you forget to check the value of the pseudo-symbol CURRENT.ERROR and continue working with meaningless values.

– The rules governing truncation and padding when assigning string variables of different lengths to each other are not taken into account.

– When the object library is switched, object modules which were loaded from the first object library are not deleted from memory. If, for example, a procedure with the name “PROC” is loaded from one object library by means of the START-PRODAMP-PROGRAM statement, “START-PRODAMP-PROGRAM PROC” will still call this procedure after switching to another object library, even if the new library contains another procedure with the same name.

Cleaning up addresses

In addressing mode 31, PRODAMP cleans up the addresses in the following cases during execution:

– if a numeric variable is used as the base for a symbolic access.

  Example

```
P := X'82CD0000';
A := P.ESTKGR0;
```

  Here, depending on the addressing mode, only X'02CD0000' or X'82CD0000' is used in the second statement to form the address.

– if the numeric variable is used as the input for the standard function LOCATION.

For DAMP, the addressing mode is a global constant that depends on the HSI. On /390 servers, DAMP uses 31-bit addressing. On x86 servers, DAMP uses 32-bit addressing. It may therefore be necessary, particularly when using PRODAMP procedures for the diagnosis of user programs, to clean up the addresses there “manually” into 24-bit addresses.

Example

```
X'887C240C' MOD X'01000000' produces X'007C240C'.
```

Passing parameters to PRODAMP

There are four ways of passing parameters to PRODAMP:

– by means of RESUME-PRODAMP-PROGRAM

– by means of START-PRODAMP-PROGRAM

– by making the appropriate entry in the procedure window, and then compiling and starting the run. In this case, the procedure should be written so that the variables which are to accept the parameters are contained in the first lines of the procedure. Then issue a MODIFY-SCREEN-LAYOUT statement to arrange the windows so that only these initial lines (with the parameter variables) are visible in the PRODAMP window. If you then overwrite the values for the parameters in this window with the current values and select the option “Go” (for “compile and go”), the procedure is recompiled and executed with the new values.

– by means of the standard PRODAMP procedure READ_WINDOW

Effects of PRODAMP on the list output

All pages in the diagnosis object which are referenced during execution of a PRODAMP procedure are regarded as referenced pages for a subsequent list output (just like pages referenced during a DAMP dialog). They are thus automatically included in a minimum list output. This feature can be used to explicitly reference in PRODAMP all the pages that are definitely to be printed.

Using EDT as a "replacement window"

The PRODAMP statements WRITE and READ, which write into and read from an EDT area, can be used to obtain pseudo-formatted dialog outputs. For this, only the following PRODAMP statements are necessary:

```
WRITE ('desired information');
....
WRITE ('desired information');
WRITE ('@COL 80 O & C'' '''); 'Truncate all lines to max. 80 characters'
WRITE ('@PRO9');              'Switch to procedure area 9'
WRITE ('@DEL');               'Delete'
WRITE ('@@PRINT1-.$VN);       'Enter PRINT statement'
WRITE ('@END');               'Switch back procedure area'
WRITE ('@DO9');               'Display the lines on the screen'
```

Truncation of the lines is necessary because the WRITE statement always pads the PRODAMP string with blanks up to the maximum length (133).

Example: guided input

```
ABC := ' '*4; XYZ := ' '*10;
WRITE ('@PRO9');
WRITE ('@DEL');
WRITE ( '@@CREATE1READ''PLEASE ENTER ABC''' );
WRITE ('@END');
WRITE ('@DO9');
READ ( ABC );
WRITE ( '@DEL' );
WRITE ('@PRO9');
WRITE ('@DEL');
WRITE ( '@@CREATE1READ''PLEASE ENTER XYZ''' );
WRITE ('@END');
WRITE ('@DO9');
READ ( XYZ );
WRITE ( '@DEL' );
```

The WRITE “@D” statements are needed to ensure that the READ statement starts at the first record of the EDT area each time. The sequence of statements shown above could be abbreviated as follows:

```
ABC := ' '*4; XYZ := ' '*10;
WRITE ('@PRO9');
WRITE ('@D');
WRITE ( '@@CREATE1READ''PLEASE ENTER ABC''' );
WRITE ( '@@CREATE2READ''PLEASE ENTER XYZ''' );
WRITE ('@END');
WRITE ('@DO9');
READ ( ABC );
READ ( XYZ );
WRITE ( '@D' );
```

Reading via READ always uses the actual length of the EDT record. The remaining characters of the string variable remain unchanged.

In addition, the string variables should always be initialized, since the default length of 133 will otherwise be assumed.

List output via the COMMAND statement

Any DAMP statements issued via COMMAND must be formulated as if they had been entered in batch or procedure mode.
This is especially important for printing lists. For this reason, the required specifications must be entered by means of the ADD-LIST-OBJECTS statement.

Example

```
COMMAND ('START-LIST-GENERATION');         "Switch to LIST mode"
COMMAND ('ADD-LIST-OBJECTS GLOBAL=OVERVIEW');
COMMAND ('ADD-LIST-OBJECTS TASK=(C''UCON'')');
COMMAND ('PRINT-LIST');
```

Calling private procedures

A PRODAMP object stored in the selected object library can be started directly by means of the DAMP statement:

```
START-PRODAMP-PROGRAME procname, PAR=(par1, par2, ...)
```

In this case, it is not necessary to know where the source module is stored or to assign a PRODAMP window. “procname” is the element name of the object module to be executed. The element with the highest existing version number is always executed. The optional parameters may be specified numerically, as decimal or hexadecimal numbers, or as strings enclosed within single quotes. The parameter types and the order in which they are specified must match the specifications in the parameter area defined within the procedure.

Example

```
ARRANGE
.P1 : RELATIVE=0,LENGTH=4,TYPE=NUMERIC;
.P2 : RELATIVE=4,LENGTH=1,TYPE=STRING;
.P3 : RELATIVE=5,LENGTH=4,TYPE=NUMERIC;
.P4 : RELATIVE=9,LENGTH=9,TYPE=STRING;
END ARRANGE;
N := PARAMETER.P1;
IF 'X' = PARAMETER.P2 THEN
....
END IF;
```

A procedure which uses the above parameters must therefore be called via a statement like this:

```
START-PRODAMP-PROGRAM procnam, PAR = (1234,'Z ',X 'AEFF','ABCDEFGHI')
```

This assigns the value 1234 to parameter P1, the letter “Z” to parameter P2, the hexadecimal number X'AEFF' to P3, and the string 'ABCDEFGHI' to parameter P4.

The parameters must be specified consecutively in the ARRANGE statement and must not be aligned. Each numeric parameter (decimal or hexadecimal) specified in the START-PRODAMP-PROGRAM statement is placed right-justified in a 4-byte field.

Numeric values can also be interpreted as bit patterns or as hexadecimal strings within the procedure.

Execution of a DAMP statement causes the entire input string to be converted into uppercase letters. This also applies to the parameters.

Interrupting private procedures

The INTERRUPT (see page 246) and RETURN (see page 246) statements can be used to interrupt execution of a procedure.

For both statements it is possible to specify a window which appears in the DAMP screen mask whenever there is an interrupt. If no window is specified, the current window appears.

RESUME-PRODAMP-PROGRAM resumes execution of the interrupted procedure at the place where it was interrupted. If no procedure was active, the procedure loaded in the PRODAMP window is started from the beginning.

Detecting and recovering execution errors

If the PRODAMP interpreter detects an error during procedure execution, it aborts the PRODAMP procedure and outputs two error messages in the message lines of the DAMP screen. The first message contains the name of the aborted PRODAMP procedure and the number of the procedure line in which the error occurred. The second message describes the error.

Pressing the K2 key and entering the statement `INFORM-PROGRAM MSG='*CANCEL'` can be used to provoke a runtime error. This makes it possible, for example, to terminate an endless loop and output a message indicating the name of the procedure and the number of the error line.

PRODAMP supports two facilities for error diagnosis: tracing (see page 247) and variable monitoring (see page 244).

Archiving private procedures

PRODAMP source and object modules can be stored in PLAM libraries as members of type S or C and loaded from these libraries when needed. Modules addressed using the ENTER_MODULE procedure are expected to be of the element type R.
Unless otherwise specified, the same library is used for all types. This library has the name SYS.USRDMP.DAMP.<ver>.
Element types C and R must be contained in the same library.

Other libraries can be specified by means of the DAMP statement ASSIGN-PRODAMP-LIBRARIES:

```
ASSIGN-PRODAMP-LIBRARIES [SOURCE=source-lib] [,OBJECT=object-lib]
```

or

```
ASSIGN-PRODAMP-LIBRARIES SOURCE=liname, OBJECT=*SOURCE
```

The specified libraries are then assigned for the desired types (if OBJECT=*SOURCE is specified, the library is assigned for all types). To switch back to the standard library, you may specify *STD instead of SYS.USRDMP.DAMP.<ver>. This switches to the library set in the user options (see “Setting user options”). SHOW-PRODAMP-LIBRARIES causes the current PRODAMP library assignments to be displayed.

PRODAMP source modules are saved and loaded exclusively via the PRODAMP window, namely by entering the appropriate code in the “Mode” field of the header line (see page 299).

Mode=Wrt (Write); the current source module is written either to the library selected by means of ASSIGN-PRODAMP-LIBRARIES or to the default library.

Mode=Rea (Read); an element from the currently selected source module library is read.

Mode=Upd (Update); the specified element is overwritten.

Mode=Inf (Inform); a list of the existing source members in the currently selected library is displayed.

PRODAMP object modules can be created by way of the PRODAMP window, but they can be loaded (and simultaneously started) only via the START-PRODAMP-PROGRAM statement.

The code “Sav” is used in the “Mode” field of the PRODAMP window in order to save an object module (after successful compilation):

Mode=Sav (Save); a PRODAMP object module is saved in the selected object library.

Examples

The following examples are concrete applications of the diagnostic language. Each example illustrates a particular aspect of the language.

Example 1: HEX calculator

This very simple example implements a hexadecimal calculator with the aid of PRODAMP. A MODIFY-SCREEN-LAYOUT statement should be entered beforehand to ensure that the first line of the PRODAMP window is visible on the screen. Modifying the expression and entering the option “Go” in the header line causes the “result” to be displayed in both hexadecimal and decimal form in line 2 of the DAMP screen.
The object being diagnosed is not accessed in this example.

```
A := X'14' + X'3B' * 24 ;       "ENTER THE DESIRED CALCULATION HERE"
MESSAGE ('RESULT HEX: '+HEX_STRING(A)+', DEC: '+DEC_STRING(A) );
```

Example 2: Searching the TFT chain of the current task

The following procedure searches through the TFT chain of the current task. Each TFT is output in window 4 in the format of the TFT DSECT. The user can then page forward to the next TFT with the aid of RESUME, or other DAMP statements can be issued.

```
IF CURRENT.TID = 0 THEN
  MESSAGE ( 'No TID/TSN given' );
  RETURN;
  END IF;
TFT := .ETCBTFT; ------------------------------------------------------------  (1)
IF TFT = 0 THEN
  MESSAGE ( 'No TFT found' );
  RETURN;
  END IF;
RET_WND := CURRENT.WNDNO; ---------------------------------------------------  (2)
ARRANGE
  WINDOW: NUMBER=4,DSECT='IDMTFT',NAME='IDMFRLNK'; ----------------------  (3)
END ARRANGE;
WHILE TFT <> 0 DO
  ARRANGE
  WINDOW: NUMBER =4,ADDRESS = TFT; --------------------------------------  (4)
END ARRANGE;
  INTERRUPT; ------------------------------------------------------------  (5)
TFT := TFT.IDMFRLNK;
  END WHILE;
RETURN WINDOW = RET_WND; ----------------------------------------------------  (6)
```

(1) The anchor of the TFT chain is in ETCBTFT. Since DAMP can localize the TFT automatically, it is not necessary to specify a base address. The task specified by TID or TSN in the PRODAMP window is taken.

(2) The number of the current window (usually the PRODAMP window) is saved so that it can be displayed again later as the topmost window.

(3) The settings for window 4 which do not change during execution are declared outside the loop. The NAME is specified because the TFT DSECT begins with an EQU * statement, which would cause the first field to be split.

(4) Only the address for the window is redefined within the loop. All other settings (including the number) are fixed in the first ARRANGE statement and remain unchanged. Interrogation of CURRENT.ERROR was omitted in this example because an unallocated memory area will automatically result in output of the error message “Requested memory area not accessible”. By default, the current task is displayed (i.e. the task set in the PRODAMP window). It is possible to specify the additional information TID=CURRENT.TID, but this is redundant.

(5) INTERRUPT interrupts the procedure and displays the current window on the screen. Due to the ARRANGE statement, window 4 is the topmost window in this display. The user can resume procedure execution by issuing a RESUME statement, causing the next TFT to be displayed.

(6) After output of the last TFT, control is returned to the window which was saved at the beginning of the procedure.

Example 3: Changing the current task

This example shows how you can avoid scrolling “endlessly” in the status window to find a task with a specific characteristic when analyzing a SLED file. It illustrates how to use PRODAMP to search for tasks which have generated a system dump and to make each of these tasks (one at a time) the current task. In other words, after execution of the procedure, the DAMP status window (window 2) is positioned such that the PCB chain of the selected task is displayed.

```
TASK := 0;
WHILE CURRENT.ERROR = 0 DO ──────────────────────────────────────────────────  (1)
NEW_TASK ( TASK );
  IF CURRENT.ERROR=0 THEN
    IF .ETCBCDSY <> 0 THEN ──────────────────────────────────────────────────  (2)
      ARRANGE WINDOW: NUMBER = 2,TID=TASK; END ARRANGE; ─────────────────────  (3)
      INTERRUPT ;
    END IF;
  END IF;
TASK := TASK + 1; ───────────────────────────────────────────────────────────  (4)
END WHILE;
```

(1) Since the NEW_TASK procedure sets CURRENT.ERROR when no more tasks can be found, this is the criterion for terminating the loop for all active tasks.

(2) The ETCBCDSY field contains the number of system dump requests for the task. This field is a TCB field and can thus be localized automatically by DAMP. To this end, DAMP uses the TCB of the current task, which is set correctly by NEW_TASK.

(3) An ARRANGE statement for window 2 with an ITN specification positions the window to the entry point for this task.

(4) For the scan, the TID must be incremented by 1. NEW_TASK then returns the next active task.

Example 4: Outputting memory areas to SYSLST

This example shows how the standard procedures DUMP_MEMORY and SET_HEADER can be used to output any desired memory areas to SYSLST.

```
TFT@ := .ETCBTFT;
WHILE TFT@ <> 0 DO
  P2FCB@ := TFT@.IDMP2FL;
  IF P2FCB@ <> 0 THEN
    SET_HEADER ( '*** P2-FCB FOR FILE '+TFT@.IDMFILE+' ****', 2, 10);
    DUMP_MEMORY ( P2FCB@, 0, LENGTH( 'ID2FCB','DS' ) );
  END IF;
  TFT@ := TFT@.IDMFRLNK;
END WHILE;
```

This procedure lists the P2-FCB of each open file.

5.7.5 Syntax diagrams

All permissible PRODAMP constructions can be determined with the aid of the syntax diagrams. On the other hand, not all constructions possible with the syntax diagrams are permissible, since type compatibility and possible restrictions with respect to names must also be taken into account. However, strictly speaking, these are not syntactical characteristics, since an expression which is illegal due, for example, to a type incompatibility can be made acceptable by choosing other designators.

In order to keep the size of the diagrams within reasonable limits, the following convention applies: connecting lines between boxes represent separators (see page 229). A separator may be omitted only before or after a special character. Separators must not be used in diagrams whose headers are framed by double lines.

The entry point for the syntax diagrams is the term “PRODAMP procedure”. This is followed by an alphabetical list of all the terms used to define the “PRODAMP procedure”.

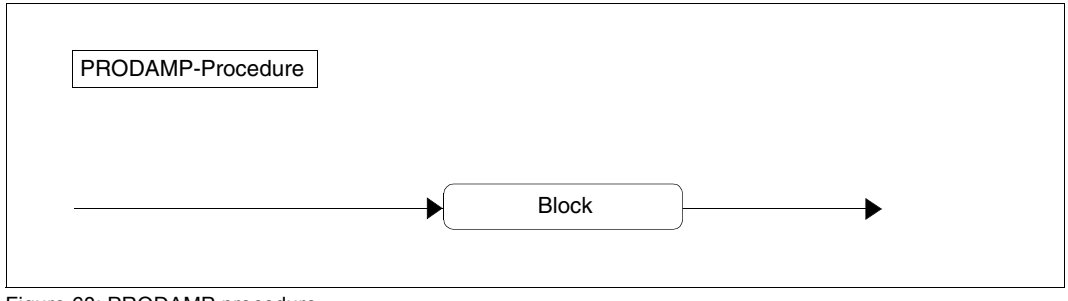


Figure 68: PRODAMP procedure

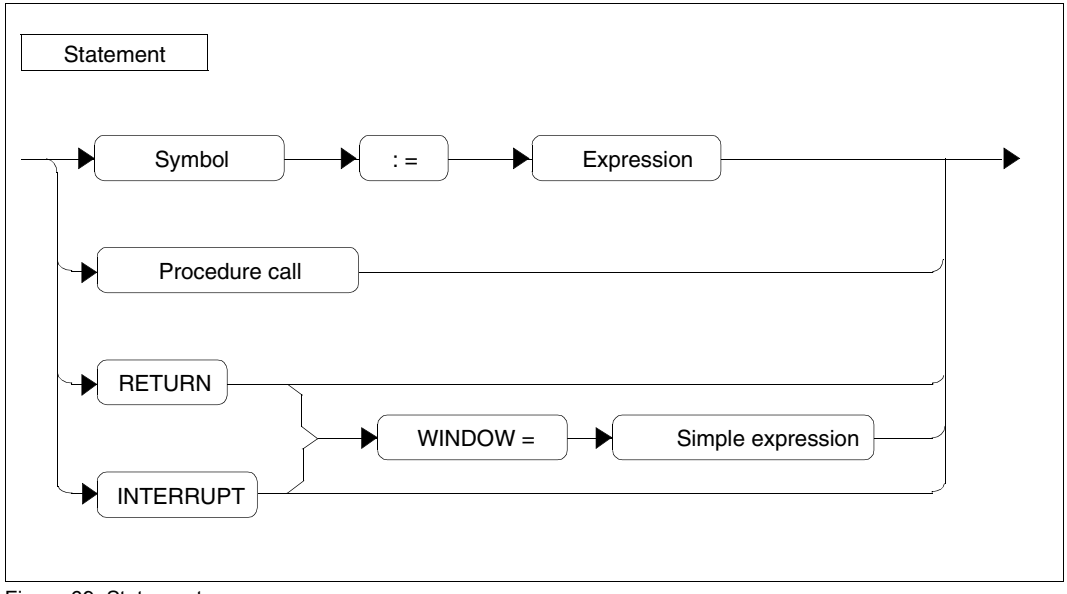


Figure 69: Statement

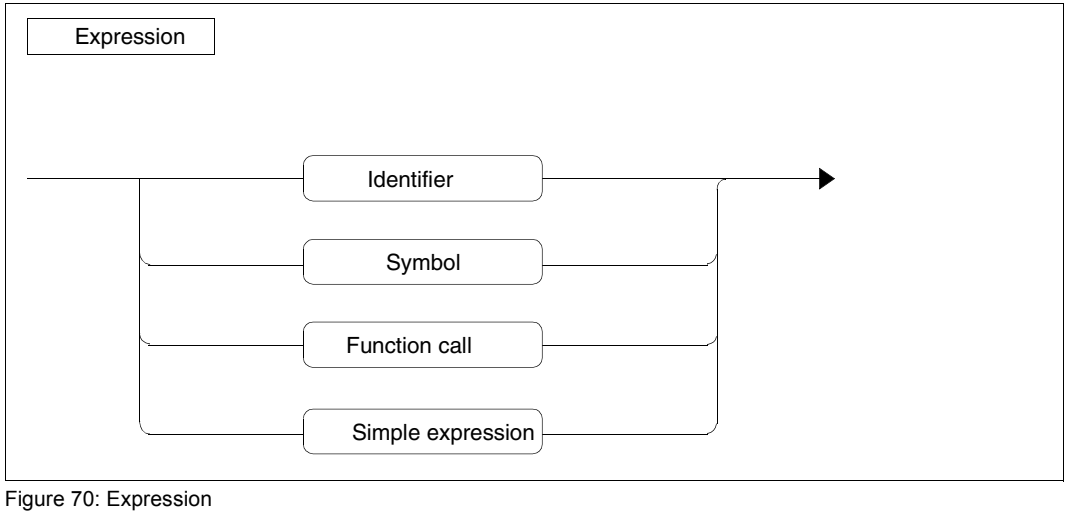


Figure 70: Expression

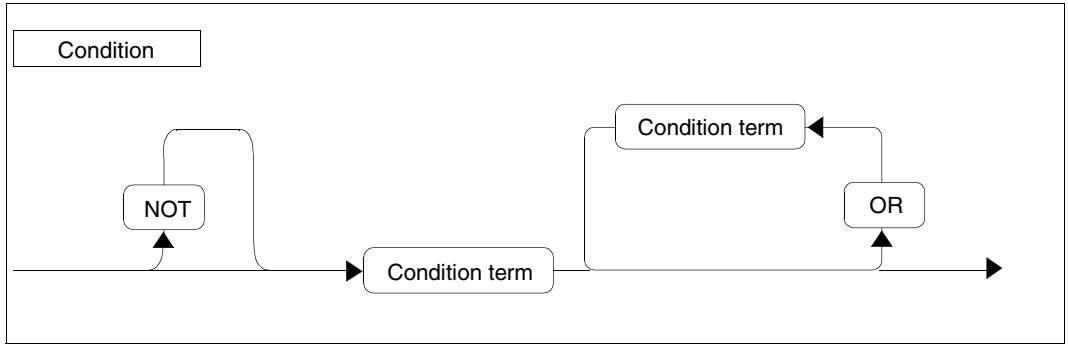


Figure 71: Condition

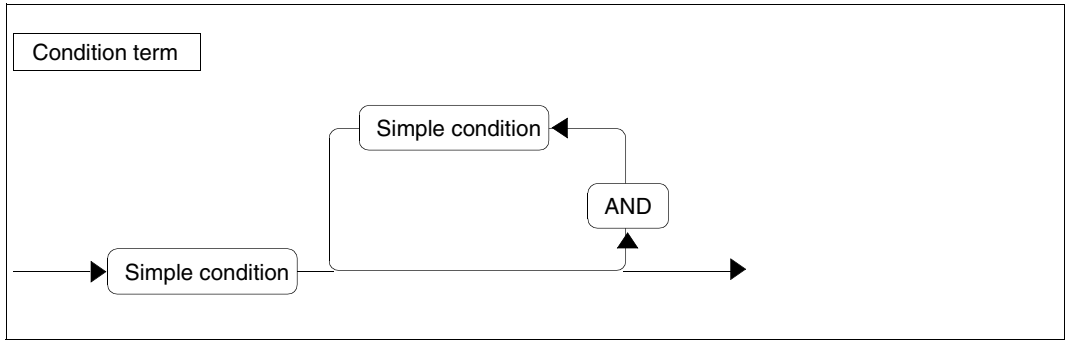


Figure 72: Condition term

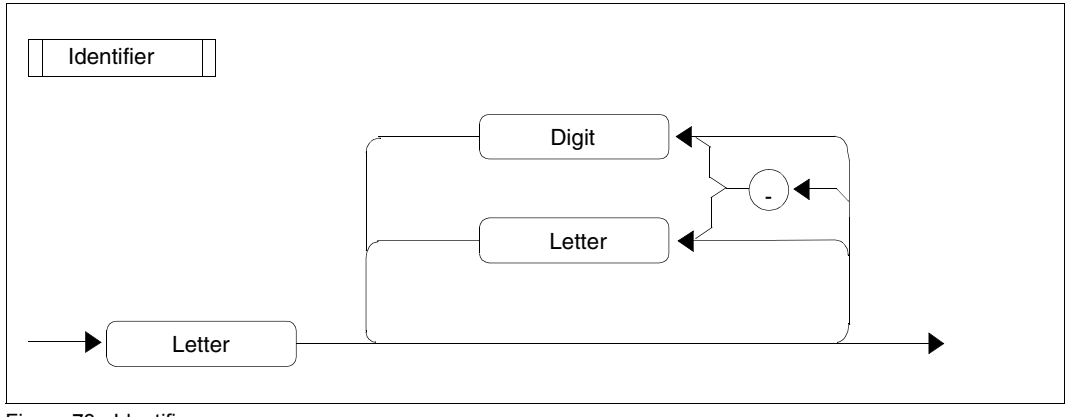


Figure 73: Identifier

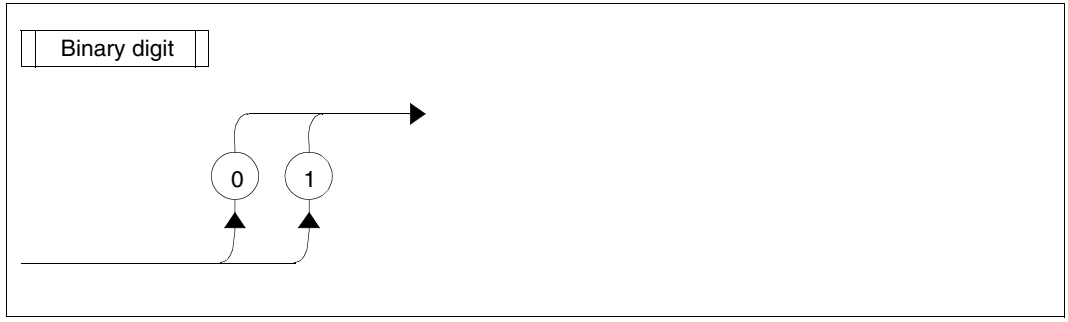


Figure 74: Binary digit

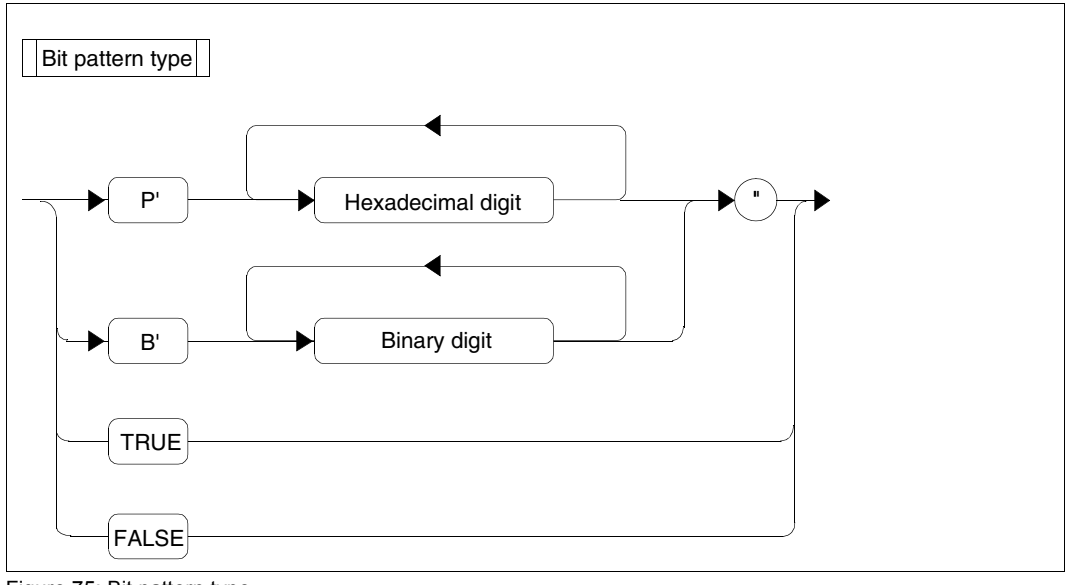


Figure 75: Bit pattern type

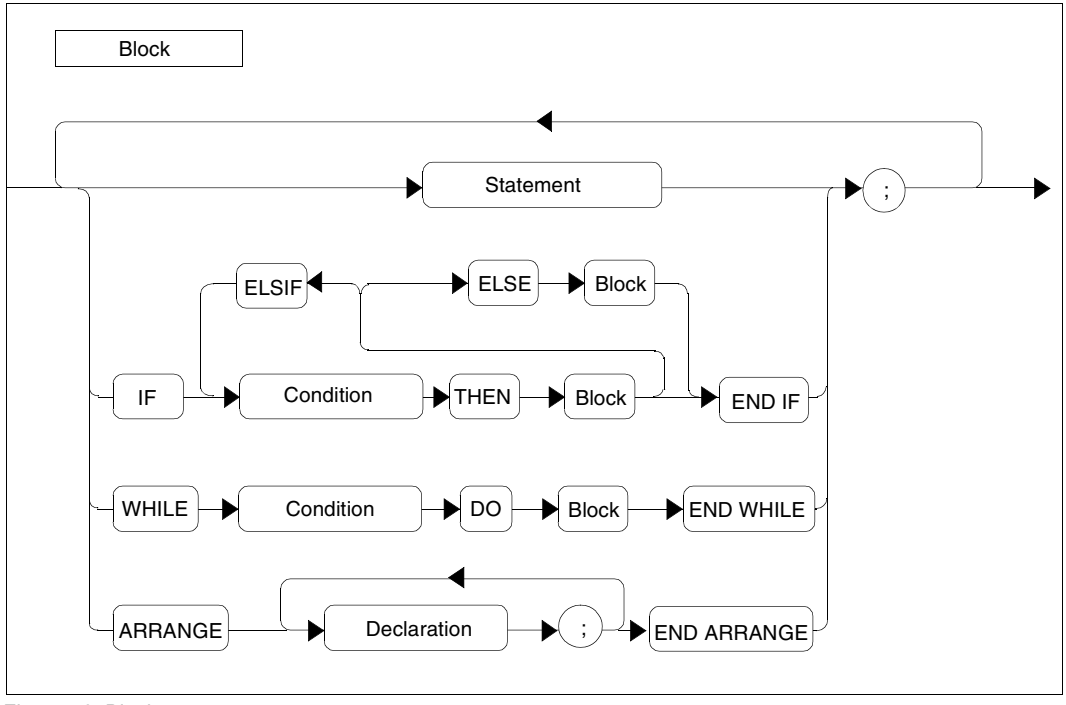


Figure 76: Block

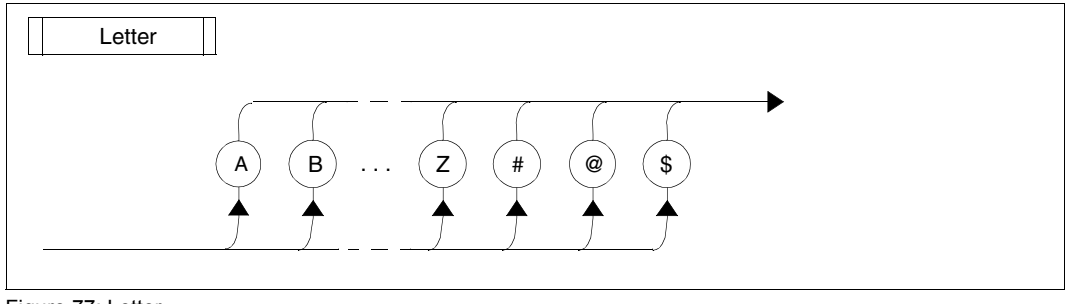


Figure 77: Letter

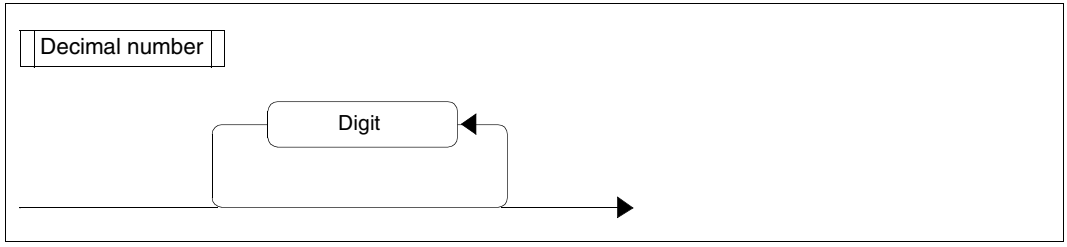


Figure 78: Decimal number

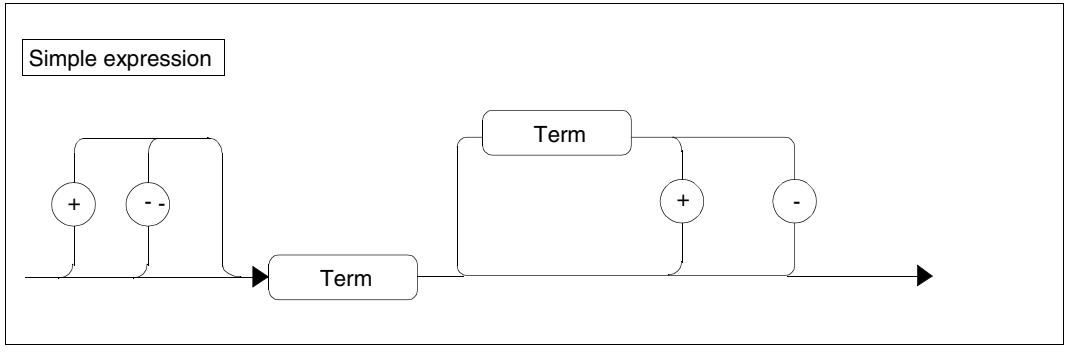


Figure 79: Simple expression

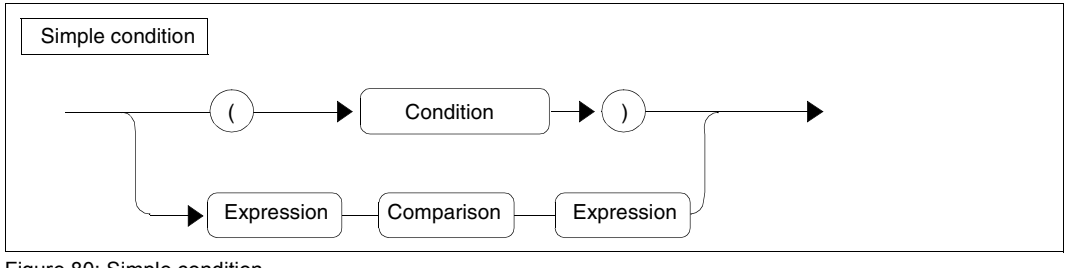


Figure 80: Simple condition

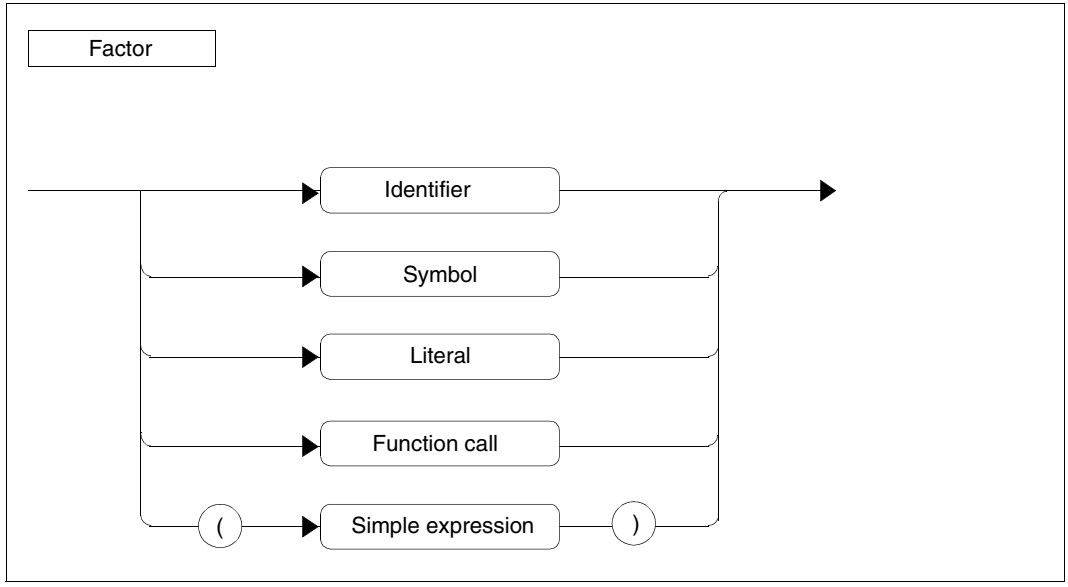


Figure 81: Factor

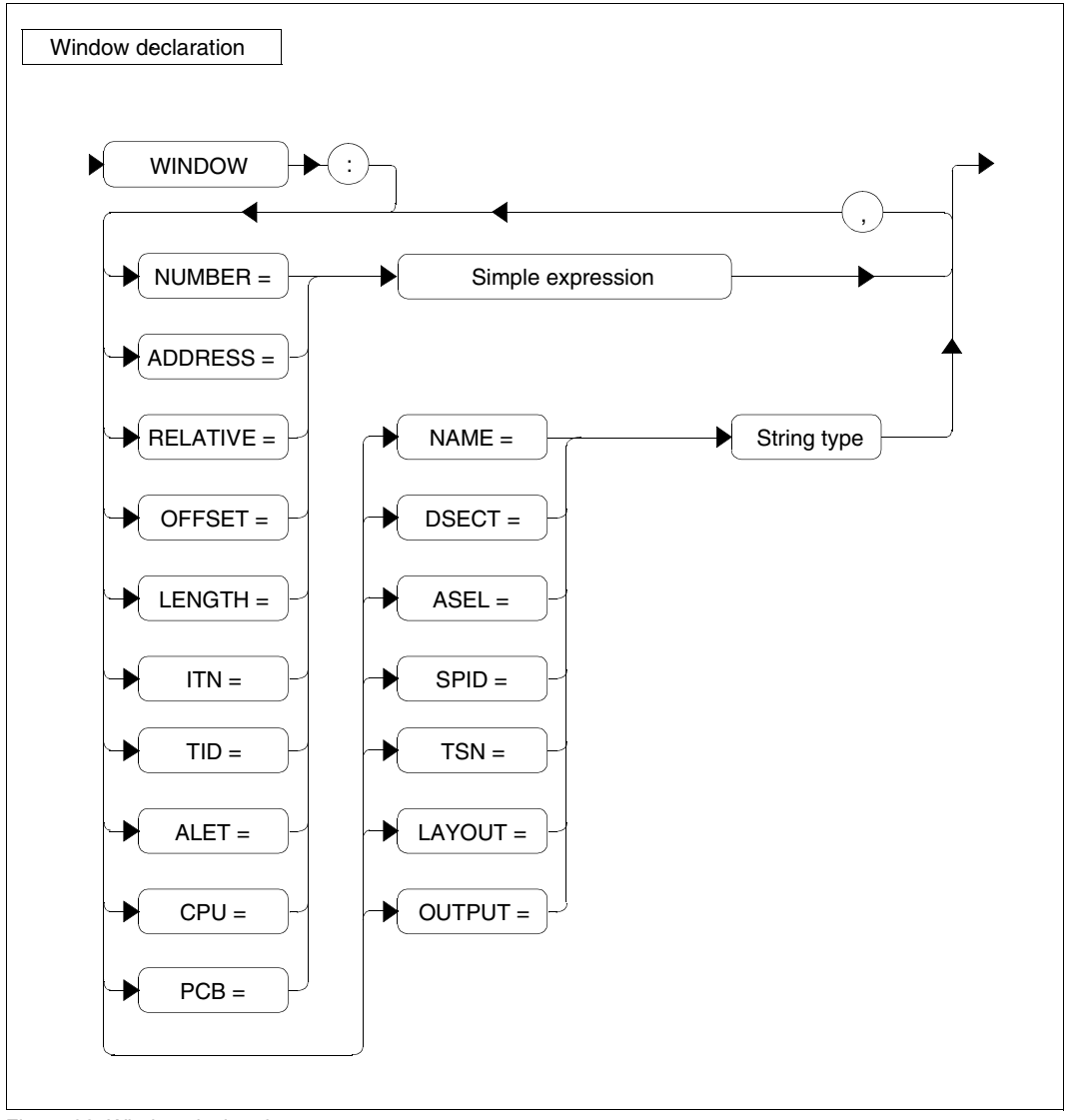


Figure 82: Window declaration

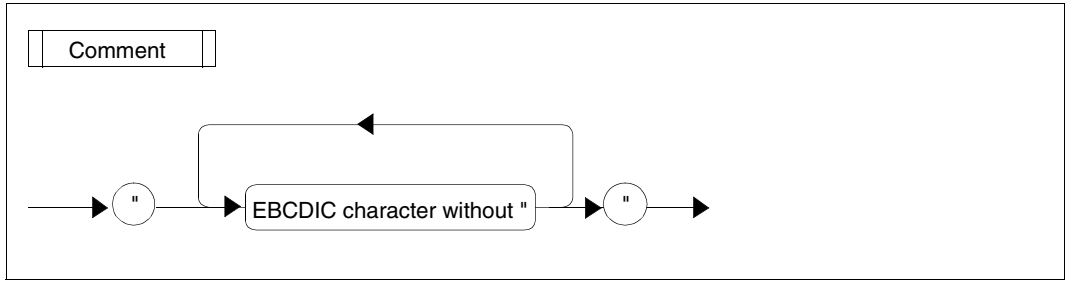


Figure 83: Comment

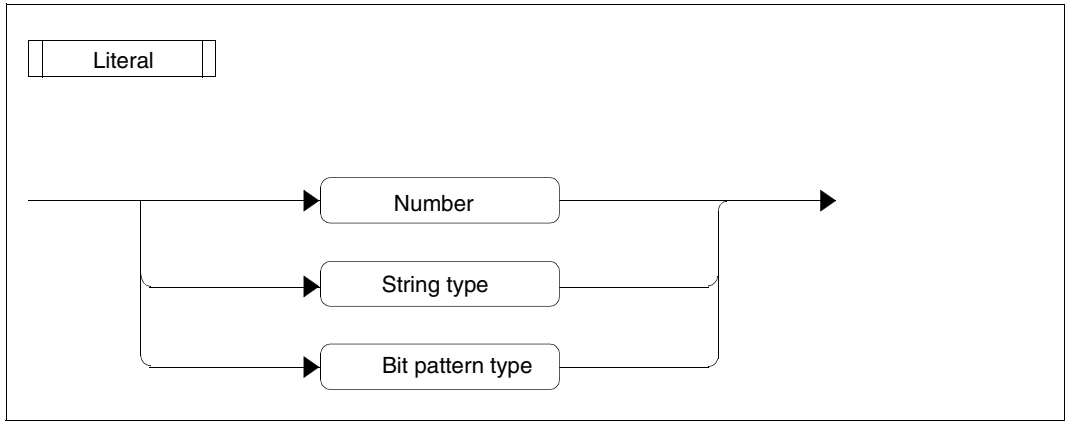


Figure 84: Literal

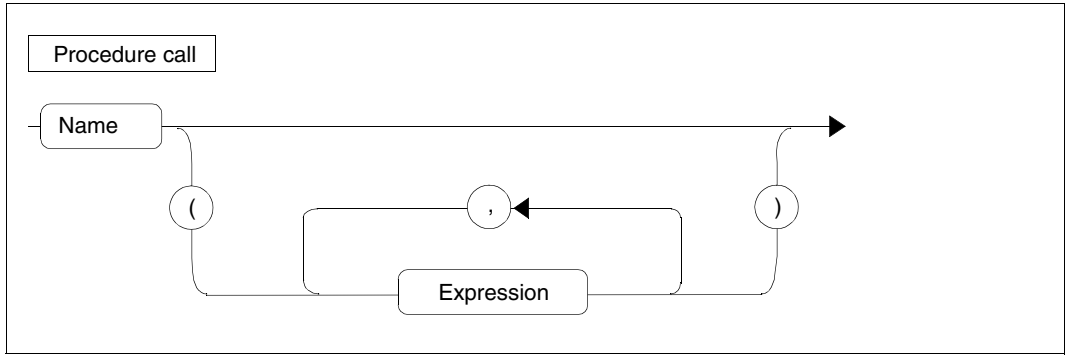


Figure 85: Procedure call

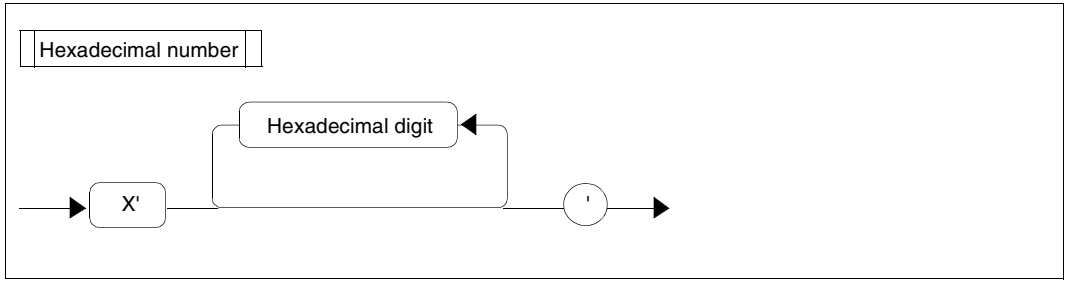


Figure 86: Hexadecimal number

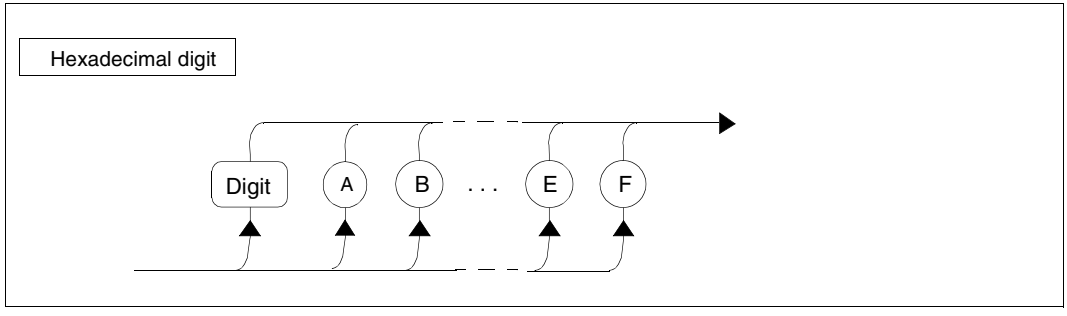


Figure 87: Hexadecimal digit

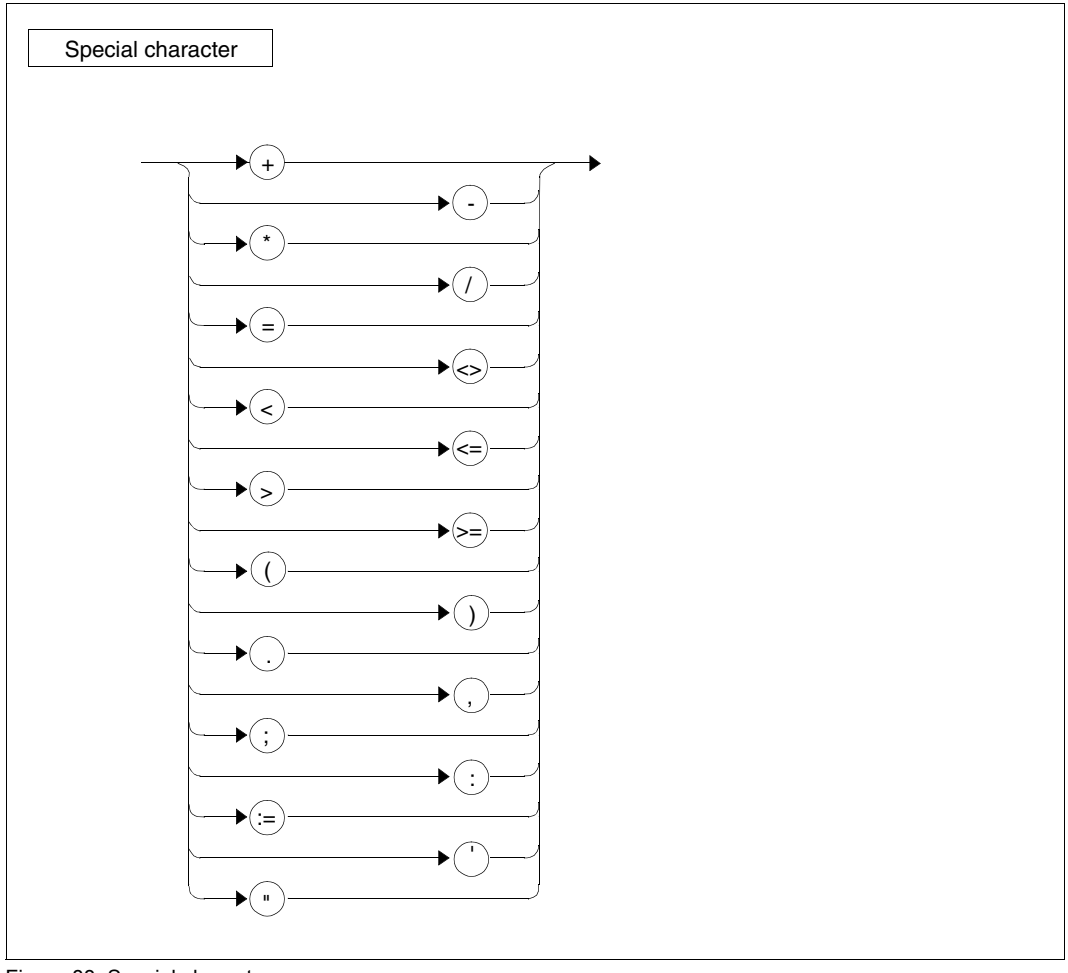


Figure 88: Special character

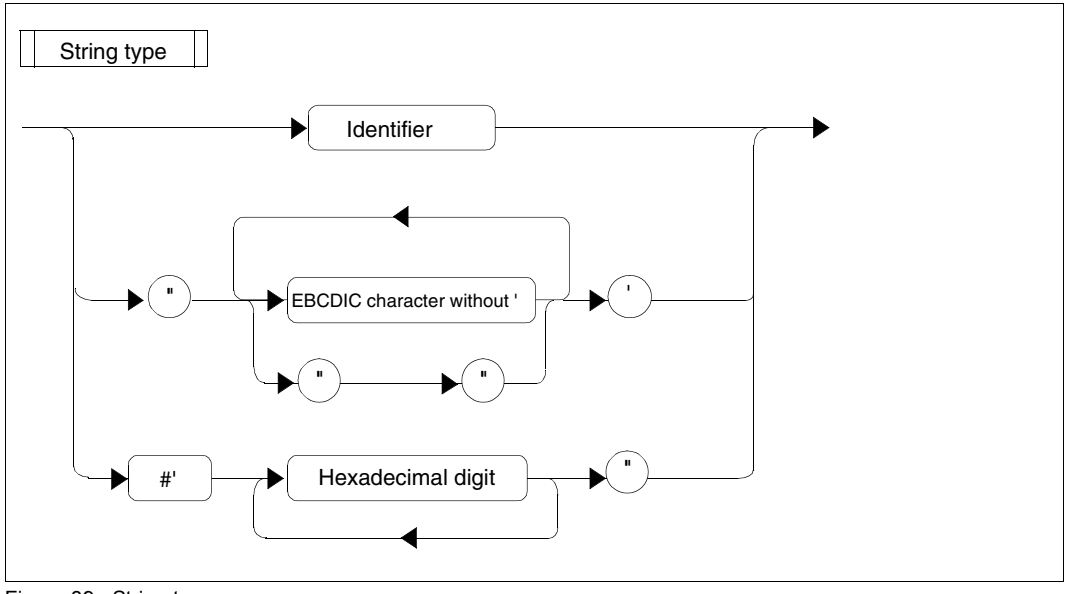


Figure 89: String type

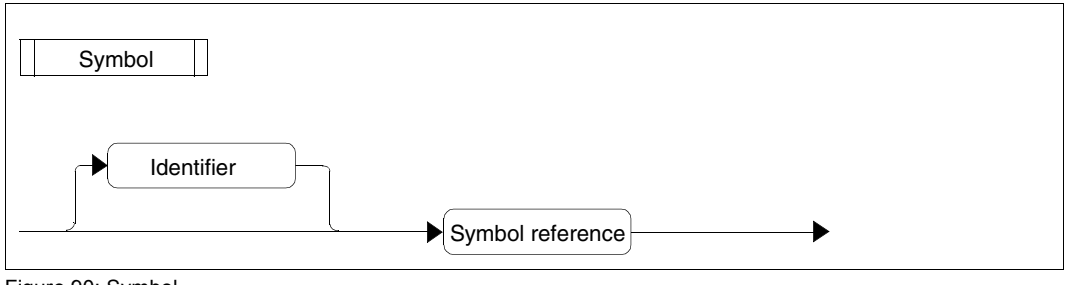


Figure 90: Symbol

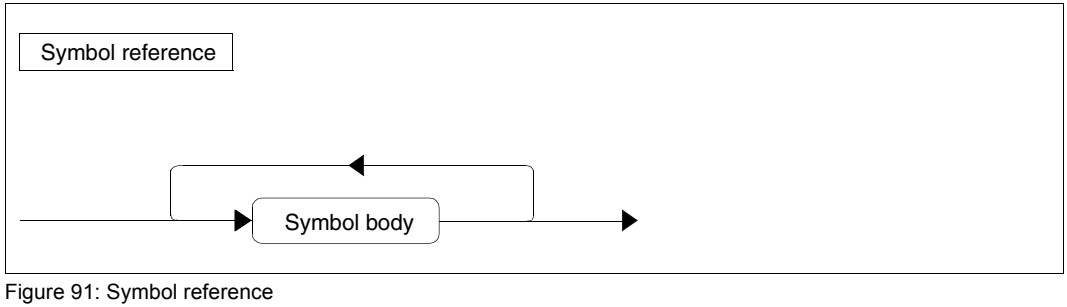


Figure 91: Symbol reference

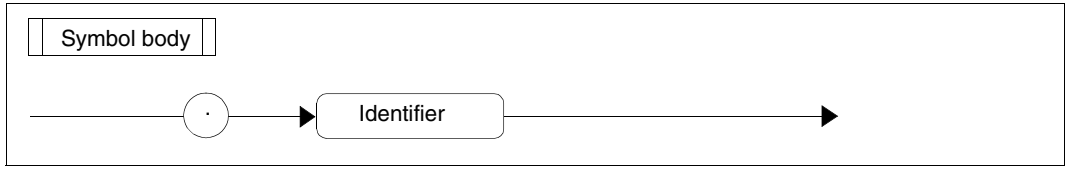


Figure 92: Symbol body

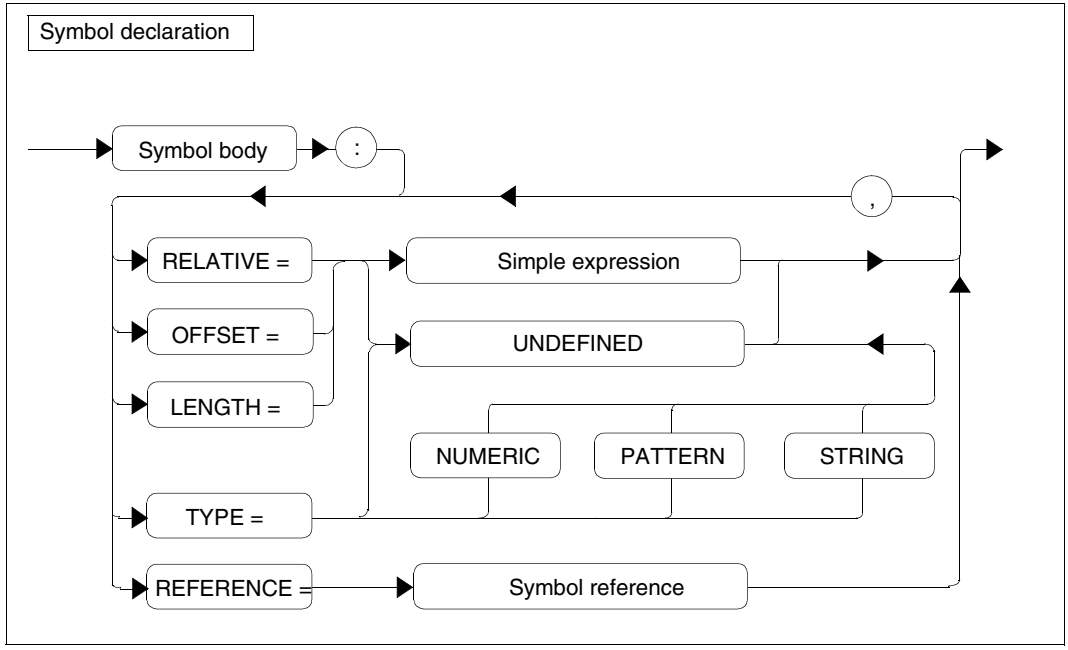


Figure 93: Symbol declaration

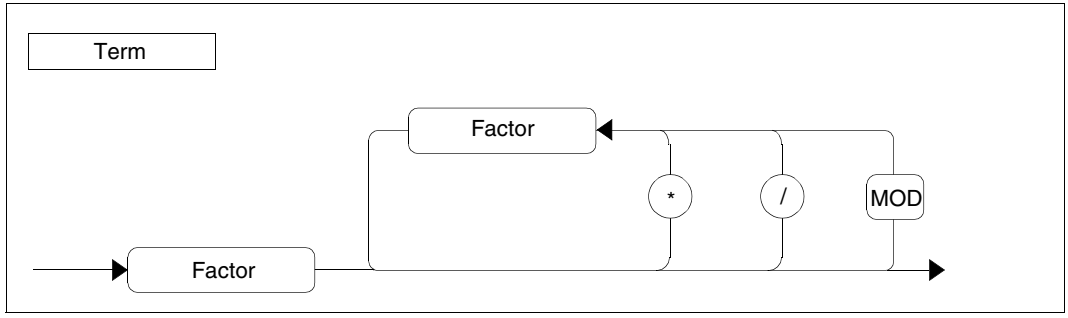


Figure 94: Term

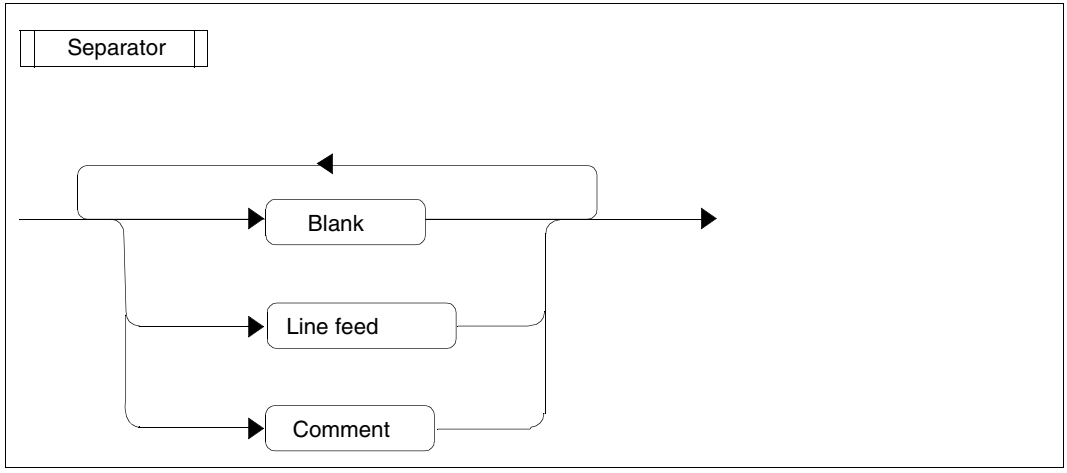


Figure 95: Separator

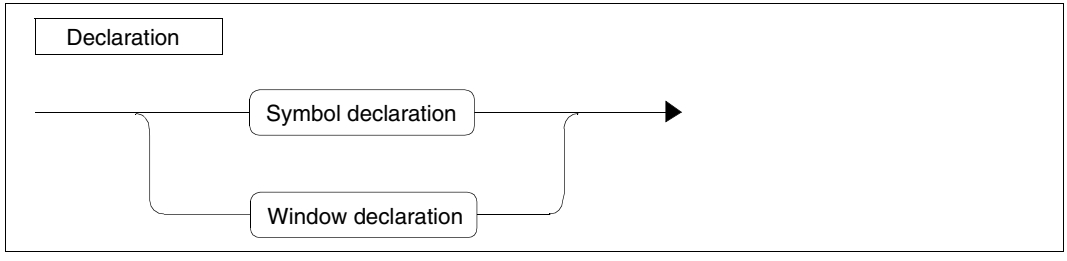


Figure 96: Declaration

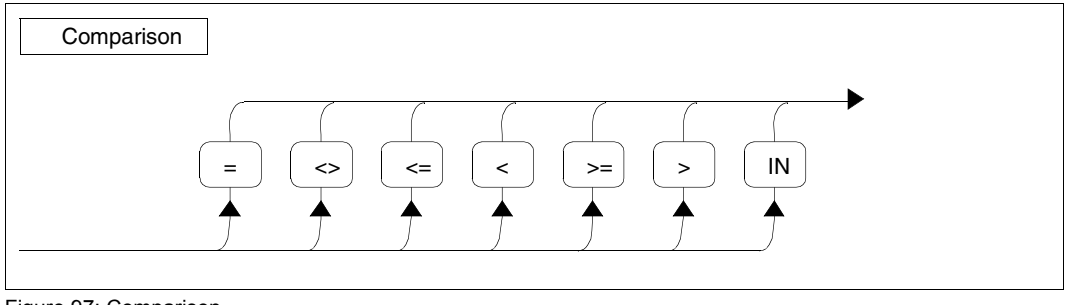


Figure 97: Comparison

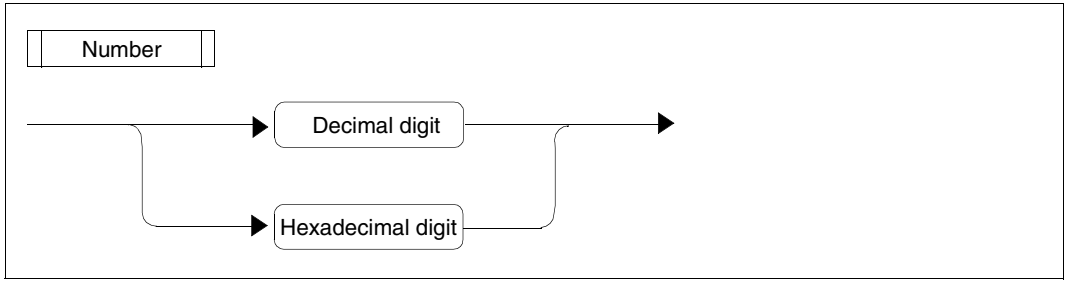


Figure 98: Number

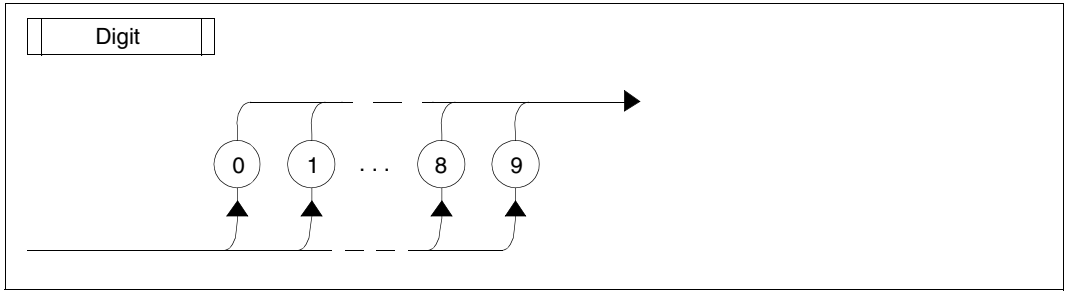


Figure 99: Digit

5.8 Software and hardware prerequisites

Installation

The following table shows all product files which are supplied with DAMP (Release Unit DAMP) and which are required when working with DAMP.

For each release item, the logical ID (for IMON), the release name, and the function are listed. The release items are contained in the installation file SYSSII.DAMP.<ver>.

Logical ID	Release name	Function
SYSSII	SYSSII.DAMP.<ver>	Installation for DAMP
SYSSDF	SYSSDF.DAMP.<ver>	SDF syntax file with the commands START-DAMP and START-DAMP-SYMBOL-GENERATOR
SYSLNK	SYSLNK.DAMP.<ver>	Dynamically loadable library of DAMP
SYSPRG	SYSPRG.DAMP.<ver>	Load program of DAMP
SYSPRG.SYMBOLS	SYSPRG.DAMP.<ver>.SYMBOLS.GEN	Symbol generator for generating private symbols
SYSMSH.D	SYSMSH.DAMP.<ver>.D	Online help German
SYSMSH.E	SYSMSH.DAMP.<ver>.E	Online help English
SYSMES	SYSMES.DAMP.<ver>	Message file. The message file is activated by DAMP.
SYSSDF.USER	SYSSDF.DAMP.<ver>.USER	User SDF syntax file with the DAMP statements. The syntax file is activated by DAMP.
SYSSMB	SYSSMB.DAMP.<ver>	Supplied library with DAMP/BS2000 symbols. Only for delivery purposes and not to be directly used. See Note 3 below.
SYSDMP	SYSDMP.DAMP.<ver>	Supplied library with DAMP-PRODAMP programs. This library contains, among other things, the PRODAMP programs for prediagnosis. The file is only needed for delivery purposes and is not to be directly used. See Note 3 below.

Furthermore the following two files are essential for working with DAMP (see Note 3 below).

File name	Function
$TSOS.SYSSMB.DAMP	Symbol system library
$TSOS.SYSDMP.DAMP	PRODAMP system library

Notes

1. The SYSPRG.xxx files are also contained as C type elements in the SYSLNK library, from which they are started using the START command. They are also supplied in file format for compatibility reasons.
2. DAMP uses the access method ANITA to access dump files and the active system. This access method must be installed correctly.
3. Note on the symbol system library and PRODAMP system library:
   Under the default setting (SYSSMB=*STD in the OPTS window), DAMP expects to find the symbols needed in order to open the diagnosis object in the file with the fixed name $TSOS.SYSSMB.DAMP (without version suffix!). On installing/updating a BS2000 system with IMON, not only the symbols required by DAMP, but also the symbols of other official products are merged into this file.

   This also applies to the PRODAMP system library in a standard installation (SYSLNK/SYSDMP=*STD in the OPTS window); the PRODAMP system library has the fixed name $TSOS.SYSDMP.DAMP.

DAMP is not coupled with any BS2000 version

See section “Performance capabilities” on page 47.

DAMP is independent of versions

See section “Performance capabilities” on page 47.

DAMP can process dump files originating from other BS2000 versions and from other BS2000 servers.

In order to evaluate dump files from BS2000 OSD/BC V11.0, the library SYSLNK.ANITA must be accessible on the system and must contain the access method ANITA V20.0A.

In order to evaluate the active system BS2000 OSD/BC V11.0 the subsystem ANITA V20.0 must be installed. In case of need it will be automatically started by DAMP.

Calling DAMP from other user IDs

The program system DAMP with the files listed above can be installed under one or more freely selectable user IDs.

In the case of private installations, it must be noted that the files are cataloged as shareable (USER-ACCESS=*ALL-USERS) and the installation user IDs (if not TSOS) are set via START-OPTION-DIALOG (see section “Modification by the user (special window: OPTIONS)” on page 133).

Other characteristics

You can use `LOAD-MODULE` to dynamically load your own analysis routines from any module libraries and start them with `START-MODULE` (see the LOAD-MODULE statement on page 191).

Users may set their own default values which are suitable for their applications (see “Setting user options” on page 133).

Prerequisites for access to the active system

Test privileges are required in order to access information in the active system.

They must be defined by the system administrator via the following command:

```
ADD-USER USER-IDENTIFICATION=userid,...,
TEST-OPTIONS=*PARAMETERS(READ-PRIVILEGE=8,
WRITE-PRIVILEGE=1[,MODIFICATION=*CONTROLLED])
```

Before DAMP is called, the test privileges must be activated using the following command:

```
MODIFY-TEST-OPTIONS PRIVILEGE=*PARAMETERS(READ=8,WRITE=1)
```

Supported terminals

The 9750 emulations of PCs are supported.

You are not allowed to use KPAC=4 under OMNIS if DAMP is running on one of the partners, since DAMP uses the K4 sequence for its own purposes.

5.9 List of DSECTs from the standard symbol files

The symbols required to analyze a BS2000 memory dump or the active BS2000 system are listed below. These symbols must be available as LMS elements of type X in the symbol library used on invocation.

By default, DAMP expects the symbols in the system symbol library. In the case of a standard installation of DAMP V4.7, this is the file with the fixed path name $TSOS.SYSSMB.DAMP.

LMS		Required for	
Element name	**Version**	**BS2000 OSD/BCVersion**	**HSI**
BS2000	180	18.0A / V9.0A	/390
BS2000-USER	180	18.0A / V9.0A	/390
XA2000	180	18.0A / V9.0A	x86
XA2000-USER	180	18.0A / V9.0A	x86
BS2000	190	19.0A / V10.0A	/390
BS2000-USER	190	19.0A / V10.0A	/390
XA2000	190	19.0A / V10.0A	x86
XA2000-USER	190	19.0A / V10.0A	x86
BS2000	200	20.0A / V11.0A	/390
BS2000-USER	200	20.0A / V11.0A	/390
XA2000	200	20.0A / V11.0A	x86
XA2000-USER	200	20.0A / V11.0A	x86
STATUS	*	Depends on the type and version	
NSDI0	*	of the DUMP generator	

Table 15: DAMP symbols in the $TSOS.SYSSMB.DAMP library

The elements STATUS and NSDI0 are only required for special cases, but should be present in the following versions in the symbol library:

STATUS: 000/ ... /011
NSDI0: 180/190/200

The differentiation between BS2000 elements, and XA2000 elements is due to HSI-specific differences in a number of hardware-based DSECTs.

A BS2000 and a BS2000-USER element (or XA2000 and XA2000-USER elements) are also required (in addition to STATUS and NSDI0).

On opening a diagnosis object with the default method (i.e. without entries or with defaults for SYMBOLS in OPEN-DIAGNOSIS-OBJECT), the required symbols are automatically loaded by DAMP. The DSECTs and SPL structures listed below are then available for the analysis in DAMP windows and in PRODAMP programs.

DSECTs shown with their names printed in bold in the list below can be automatically localized.

Elements BS2000/180 and XA2000/180 (for BS2000 V18.0 = BS2000/OSD-BC V9.0 with /390-HSI resp. x86-HSI)

Assembler DSECTs

ASAVDSSM	ASIMDBHD	ASIPUCON	DSTE	DWQE	DWQH
EBWL	ECSA	ECSE	ECSX	ECTLP	EERLWA
EGCTRAC	**EJCB**	EJTBP	EMICWA	EMMDDSMD	EMRCWA
EOLDTBLE	EORD	EPDR	ERTWA	ESOFWA	ESTK
ETCB	ETLT	ETMGPT	ETRAC	**EVSMT**	**EVUMT**
EVYVWA	**EXVT**	FPTA34	FPTE34	HCTX	IAR
IBW	IDMTFT	ID1FCB	ID1FCBE	ID2FCB	IELS
INTE	INTEMMP	INTESMP	ML	NDXMTE	NDXT
NEHX$MDL	NIOSC	NLKDEXT	NLKDHEA	NLKDSAV	NLKSJDS
NOTEDS	NRXDPL	NRXIPL	NRXPPL	NSCDH	NSCDL
NSDIO	NSDI1	NSDLPLD	NSDPPLD	NSDTPLD	NSICONFT
NSISWID	PDTHDR	PDTREC	PPTE	PSA	RECBUFF
STRCWA	TERMMSG	TLTE	TTSAVE	VATE	XD1FCB

SPL structures

BS_CTX_VECTOR_REC_MDL	CTX_VECTOR_REC_MDL
DBL_OPTIONS_COM_MDL	DBL_OPTIONS_P_C_MDL
DBL_OPTIONS_S_P_MDL	LIBRARY_TAB_D_MDL
LU_CTX_VECTOR_REC_MDL	LU_MEM_POOL_VECTOR_REC_MDL
MEM_POOL_REC_MDL	NSIVR_MDL
NTFHOOK_MDL	PBMM_ATTR_MDL
PROGRAM_LOAD_LIST_MDL	RECORD_D_MDL
RECORD_H_MDL	TABLE_D_MDL
TASK_TAB_MDL	VERSION_MDL

Elements BS2000-USER/180 and XA2000-USER/180 (additionally for BS2000 V18.0 = BS2000/OSD-BC V9.0 with /390-HSI resp. X86-HSI)

Assembler DSECTs

CSTMP	DDZCCB	DECRCOD	DECRNAM	DRPVST	DSFTB
DSHED	DSPTB	DSSTB	DSUTB	DS3BCB	DS4LBL
DS6STK	EACQ	EBVDT	ECSE	EGCARIGT	EGCMXLDS
EGCW_MDL	EGSTRAC	ENRTPL	EPDMM	EPPT	ESTK
ETCOMEV	ETCOMTBL	ETCOPRTL	ETMCH	ETMMH	GARE
IBO	ICACFCP	ICACFDP	ICACFFP	ICACFHP	ICACFMP
ICACFPP	ICAEE4	ICO	IDBCHAPL	IDBCOPPL	IDBERAPL
IDBFSTPL	IDBPFLPL	IDBPFLPX	IDBRELPL	IDCEG	IDCES
IDCEXS	IDEE3	IDJES	IDJEXT	IDJEXT2	IDKCATPL
IDPBTAPL	IDQPAMPL	IDVTS	INST	LFCB	NAR
NDVESPDS	NEHX$MDL	NERRLOCK	NLOCK	NLPT	NLWALOCK
NRTSEHDT	NSCB	NSPAPLD	NSPR	NSUBLOCK	NTIM
NTRCLOCK	NVPSPL	RKLOG	SD	SPAD	SPOD
WORDLIST	XDBFSTPL	XDPBTAPL	XDQPAMPL	XRD	

SPL structures

$JCBRW_PL_MDL	$SSMCEO_PL_MDL
$SSMENT_PL_MDL	$SSMERA_PL_MDL
ADDPLNK_MDL	CREPOOL_MDL
DELPOOL_MDL	ECSA_MDL
EAM_MDL	ESMFHDR
ESMIFID_MDL	ESMRETC_MDL
ESTK_MDL	NSIVR_MDL
PAM_MDL	REMPLNK_MDL
VERSION_MDL	

Elements BS2000/190 and XA2000/190 (for BS2000 V19.0 = BS2000 OSD/BC V10.0 with /390-HSI resp. x86-HSI)

Assembler DSECTs

ASAVDSSM	ASIMDBHD	ASIPUCON	DSTE	DWQE	DWQH
EBWL	ECSA	ECSE	ECSX	ECTLP	EERLWA
EGCTRAC	**EJCB**	EJTBP	EMICWA	EMMDDSMD	EMRCWA
EOLDTBLE	EORD	EPDR	ERTWA	ESOFWA	ESTK
ETCB	ETLT	ETMGPT	ETRAC	**EVSMT**	**EVUMT**
EVYVWA	**EXVT**	FPTA34	FPTE34	HCTX	IAR
IBW	IDMTFT	ID1FCB	ID1FCBE	ID2FCB	IELS
INTE	INTEMMP	INTESMP	ML	NDXMTE	NDXT
NEHX$MDL	NIOSC	NLKDEXT	NLKDHEA	NLKDSAV	NLKSJDS
NOTEDS	NRXDPL	NRXIPL	NRXPPL	NSCDH	NSCDL
NSDIO	NSDI1	NSDLPLD	NSDPPLD	NSDTPLD	NSICONFT
NSISWID	PDTHDR	PDTREC	PPTE	PSA	RECBUFF
STRCWA	TERMMSG	TLTE	TTSAVE	VATE	XD1FCB

SPL structures

BS_CTX_VECTOR_REC_MDL	CTX_VECTOR_REC_MDL
DBL_OPTIONS_COM_MDL	DBL_OPTIONS_P_C_MDL
DBL_OPTIONS_S_P_MDL	LIBRARY_TAB_D_MDL
LU_CTX_VECTOR_REC_MDL	LU_MEM_POOL_VECTOR_REC_MDL
MEM_POOL_REC_MDL	NSIVR_MDL
NTFHOOK_MDL	PBMM_ATTR_MDL
PROGRAM_LOAD_LIST_MDL	RECORD_D_MDL
RECORD_H_MDL	TABLE_D_MDL
TASK_TAB_MDL	VERSION_MDL

Elements BS2000-USER/190 and XA2000-USER/190 (additionally for BS2000 V19.0 = BS2000 OSD/BC V10.0 with /390-HSI resp. X86-HSI)

Assembler DSECTs

CSTMP	DDZCCB	DECRCOD	DECRNAM	DRPVST	DSFTB
DSHED	DSPTB	DSSTB	DSUTB	DS3BCB	DS4LBL
DS6STK	EACQ	EBVDT	ECSE	EGCARIGT	EGCMXLDS
EGCW_MDL	EGSTRAC	ENRTPL	EPDMM	EPPT	ESTK
ETCOMEV	ETCOMTBL	ETCOPRTL	ETMCH	ETMMH	GARE
IBO	ICACFCP	ICACFDP	ICACFFP	ICACFHP	ICACFMP
ICACFPP	ICAEE4	ICO	IDBCHAPL	IDBCOPPL	IDBERAPL
IDBFSTPL	IDBPFLPL	IDBPFLPX	IDBRELPL	IDCEG	IDCES
IDCEXS	IDEE3	IDJES	IDJEXT	IDJEXT2	IDKCATPL
IDPBTAPL	IDQPAMPL	IDVTS	INST	LFCB	NAR
NDVESPDS	NEHX$MDL	NERRLOCK	NLOCK	NLPT	NLWALOCK
NRTSEHDT	NSCB	NSPAPLD	NSPR	NSUBLOCK	NTIM
NTRCLOCK	NVPSPL	RKLOG	SD	SPAD	SPOD
WORDLIST	XDBFSTPL	XDPBTAPL	XDQPAMPL	XRD	

SPL structures

$JCBRW_PL_MDL	$SSMCEO_PL_MDL
$SSMENT_PL_MDL	$SSMERA_PL_MDL
ADDPLNK_MDL	CREPOOL_MDL
DELPOOL_MDL	ECSA_MDL
EAM_MDL	ESMFHDR
ESMIFID_MDL	ESMRETC_MDL
ESTK_MDL	NSIVR_MDL
PAM_MDL	REMPLNK_MDL
VERSION_MDL	

Elements BS2000/200 and XA2000/200 (for BS2000 V20.0 = BS2000 OSD/BC V11.0 with /390-HSI resp. x86-HSI)

Assembler DSECTs

ASAVDSSM	ASIMDBHD	ASIPUCON	DSTE	DWQE	DWQH
EBWL	ECSA	ECSE	ECSX	ECTLP	EERLWA
EGCTRAC	**EJCB**	EJTBP	EMICWA	EMMDDSMD	EMRCWA
EOLDTBLE	EORD	EPDR	ERTWA	ESOFWA	ESTK
ETCB	ETLT	ETMGPT	ETRAC	**EVSMT**	**EVUMT**
EVYVWA	**EXVT**	FPTA34	FPTE34	HCTX	IAR
IBW	IDMTFT	ID1FCB	ID1FCBE	ID2FCB	IELS
INTE	INTEMMP	INTESMP	ML	NDXMTE	NDXT
NEHX$MDL	NIOSC	NLKDEXT	NLKDHEA	NLKDSAV	NLKSJDS
NOTEDS	NRXDPL	NRXIPL	NRXPPL	NSCDH	NSCDL
NSDIO	NSDI1	NSDLPLD	NSDPPLD	NSDTPLD	NSICONFT
NSISWID	PDTHDR	PDTREC	PPTE	PSA	RECBUFF
STRCWA	TERMMSG	TLTE	TTSAVE	VATE	XD1FCB

SPL structures

BS_CTX_VECTOR_REC_MDL	CTX_VECTOR_REC_MDL
DBL_OPTIONS_COM_MDL	DBL_OPTIONS_P_C_MDL
DBL_OPTIONS_S_P_MDL	LIBRARY_TAB_D_MDL
LU_CTX_VECTOR_REC_MDL	LU_MEM_POOL_VECTOR_REC_MDL
MEM_POOL_REC_MDL	NSIVR_MDL
NTFHOOK_MDL	PBMM_ATTR_MDL
PROGRAM_LOAD_LIST_MDL	RECORD_D_MDL
RECORD_H_MDL	TABLE_D_MDL
TASK_TAB_MDL	VERSION_MDL

Elements BS2000-USER/200 and XA2000-USER/200
(additionally for BS2000 V20.0 = BS2000 OSD/BC V11.0 with /390-HSI resp. X86-HSI)

Assembler DSECTs

CSTMP	DDZCCB	DECRCOD	DECRNAM	DRPVST	DSFTB
DSHED	DSPTB	DSSTB	DSUTB	DS3BCB	DS4LBL
DS6STK	EACQ	EBVDT	ECSE	EGCARIGT	EGCMXLDS
EGCW_MDL	EGSTRAC	ENRTPL	EPDMM	EPPT	ESTK
ETCOMEV	ETCOMTBL	ETCOPRTL	ETMCH	ETMMH	GARE
IBO	ICACFCP	ICACFDP	ICACFFP	ICACFHP	ICACFMP
ICACFPP	ICAEE4	ICO	IDBCHAPL	IDBCOPPL	IDBERAPL
IDBFSTPL	IDBPFLPL	IDBPFLPX	IDBRELPL	IDCEG	IDCES
IDCEXS	IDEE3	IDJES	IDJEXT	IDJEXT2	IDKCATPL
IDPBTAPL	IDQPAMPL	IDVTS	INST	LFCB	NAR
NDVESPDS	NEHX$MDL	NERRLOCK	NLOCK	NLPT	NLWALOCK
NRTSEHDT	NSCB	NSPAPLD	NSPR	NSUBLOCK	NTIM
NTRCLOCK	NVPSPL	RKLOG	SD	SPAD	SPOD
WORDLIST	XDBFSTPL	XDPBTAPL	XDQPAMPL	XRD	

SPL structures

$JCBRW_PL_MDL	$SSMCEO_PL_MDL
$SSMENT_PL_MDL	$SSMERA_PL_MDL
ADDPLNK_MDL	CREPOOL_MDL
DELPOOL_MDL	ECSA_MDL
EAM_MDL	ESMFHDR
ESMIFID_MDL	ESMRETC_MDL
ESTK_MDL	NSIVR_MDL
PAM_MDL	REMPLNK_MDL
VERSION_MDL	

5.10 DAMP messages

The DAMP messages have the message class DMP. Information on individual messages can be obtained in ongoing operation with /HELP-MSG-INFORMATION.

6 NDMDAMP Generating diagnostic documents

NDMDAMP is a PRODAMP procedure package within DAMP that extracts and analyzes the data relevant for NDM from a SLED, system dump or the active system.

Three methods are offered to control the output scope of the dump function:

- a normal analysis,
- a comprehensive “maximum” analysis or
- a restricted analysis (tailored to each problem).

NDMDAMP can be called interactively, directly via DAMP or via predefined ENTER jobs. The method used for parameterization depends on how it is called.

6.1 Calling NDMDAMP

NDMDAMP can be called by various methods:

- interactively, via the START-NDM-DIAGNOSIS command

  The START-NDM-DIAGNOSIS command is available following the command MOD[IFY]-SDF[-OPTIONS] [$TSOS.]SYSSDF.NDMDAMP.<ver>.USER.

- from DAMP

  When called from DAMP, the standard analyses or special analyses can be selected. The settings of the options must be explicitly entered in the PRODAMP procedure NDM.

- with predefined ENTER jobs

  The LMS library SYSENT.NDMDAMP.<ver> is supplied with six ENTER jobs, which provide various analyses.

START-NDM-DIAGNOSIS
Analyze NDM data

Function

The START-NDM-DIAGNOSIS command analyzes the NDM data from the active system, a system dump or a SLED dump using NDMDAMP.

Format

START-NDM-DIAGNOSIS

OBJECT = ***SYSTEM** / <filename 1..54> / ***LINK**(...)

***LINK**(...)

LINK-NAME = <filename 1..8 without-gen>

,INFORMATION = ***STD** / ***ALL / *TRACE / *DRV / *DRV-ALL / *EXECUTION-TRACE** / ***PAR**AMETERS(...)

***PAR**AMETERS(...)

IO-CONTROL-DATA = ***STD** / ***NO / *ALL / *ALL-WITH-EXECUTION-TRACE**
,**BAVOLMON** = ***STD** / ***NO / *ALL / *ALL-WITH-EXECUTION-TRACE**
,**NKA** = ***STD** / ***NO / *ALL / *ALL-WITH-EXECUTION-TRACE**
,**NKV** = ***STD** / ***NO / *STD-WITH-EXECUTION-TRACE**
,**NKS** = ***STD** / ***NO / *ALL / *ALL-WITH-EXECUTION-TRACE**
,**NKR** = ***STD** / ***NO / *ALL / *ALL-WITH-EXECUTION-TRACE**
,**DRV** = ***STD** / ***NO / *STD-WITH-EXECUTION-TRACE**
,**TRACE** = ***STD** / ***NO / *STD-WITH-EXECUTION-TRACE**
,**SHC-OSD** = ***STD** / ***NO / *STD-WITH-EXECUTION-TRACE**

,**ENVIRONMENT** = ***STD** / ***PAR**AMETERS(...)

***PAR**AMETERS(...)

PROGRAM-NAME = ***STD** / <filename 1..54 without-gen>
,SYMBOL-LIBRARY = ***STD** / <filename 1..54 without-gen>
,PRODAMP-LIBRARY = ***STD** / <filename 1..54 without-gen>
,NUMBER-OF-RESTARTS = **1** / <integer 0..5>
,OUTPUT = ***STD** / ***SYSLST** / <filename 1..54>

Operands

OBJECT = *SYSTEM / <filename 1..54> / *LINK(...)
Specifies which dump file is to be analyzed.

OBJECT = *LINK(...)

LINK-NAME = <filename 1..8 without-gen>
The dump file is specified via its link name.

INFORMATION =
Specifies the scope of the dump analysis.

INFORMATION = *STD
Maximum analysis, without EXPDT, without task-local data of NKA and NKS, and without NKR modules.

INFORMATION = *ALL
Maximum analysis.

INFORMATION = *TRACE
Only NDM traces.

INFORMATION = *DRV
Only DRV and NDM trace.

INFORMATION = *DRV-ALL
NKA, NKV, DRV, IO-CONTROL without EXPDT, BAVOLMON and NDM traces.

INFORMATION = *EXECUTION-TRACE
Maximum analysis with PRODAMP trace enabled. EDT must be available for this purpose.

INFORMATION = *PARAMETERS(...)

IO-CONTROL-DATA = *STD / *NO / *ALL / *ALL-WITH-EXECUTION-TRACE
Analysis of IO-CONTROL data. For *ALL, the EXPDT data module is also output.

BAVOLMON = *STD / *NO / *ALL / *ALL-WITH-EXECUTION-TRACE
Analysis of basic volume monitoring. For *ALL, additional information for NDIVT is output.

NKA = *STD / *NO / *ALL / *ALL-WITH-EXECUTION-TRACE
Analysis of NKALLOC (NDM allocator). For *ALL, additional task-local data is output.

NKV = *STD / *NO / *STD-WITH-EXECUTION-TRACE
Analysis of NKVMOUNT (NDM volume monitoring).

NKS = *STD / *NO / *ALL / *ALL-WITH-EXECUTION-TRACE
Analysis of NKSECRES (NDM Secure). For *ALL, additional task-local data is output.

NKR = *STD / *NO / *ALL / *ALL-WITH-EXECUTION-TRACE
Analysis of NKRECONF (NDM reconfiguration administration).For *ALL, the NKR are also output.

DRV = *STD / *NO / *STD-WITH-EXECUTION-TRACE
Analysis of DRV (Dual Recording by Volume).

TRACE = *STD / *NO / *STD-WITH-EXECUTION-TRACE
Analysis of the NDM trace.

SHC-OSD = *STD / *NO / *STD-WITH-EXECUTION-TRACE
Analysis of the SHC-OSD (Storage Host Component for BS2000).

A dump file from a system with an active subsystem SHC-OSD can be analyzed correctly (i.e. without errors) with NDMDAMP on a system without SHC-OSD only if NDMDAMP is started as follows:

```
START-NDM-DIAGNOSIS OBJECT=..., INF = *PAR(SHC-OSD=*NO)
```

ENVIRONMENT = *STD / *PARAMETERS(...)
Defines the dump environment.

ENVIRONMENT = *PARAMETERS(...)

PROGRAM-NAME = *STD / <filename 1..54 without-gen>
Specifies the DAMP file.
For *STD, /START-DAMP is used.

SYMBOL-LIBRARY = *STD / <filename 1..54 without-gen>
Name of the symbol library.
For *STD, the system library of DAMP is used: SYSSMB.DAMP.

PRODAMP-LIBRARY = *STD / <filename 1..54 without-gen>
Name of the PRODAMP library with NDMDAMP.
For *STD, the system library of DAMP is used: SYSDMP.DAMP.

NUMBER-OF-RESTARTS = 1 / <integer 0..5>
Number of attempts to be made to restart with the next procedure following the occurrence of errors in NDMDAMP. If a value greater than 0 is specified, EDT must be available.

OUTPUT = *STD / *SYSLST / <filename 1..54>
Name of the output file.
For *STD, the file SYSLST.NDMDAMP.<date>.<time> is used.

6.1.1 Calling NDMDAMP from DAMP

When NDMDAMP is called from DAMP, the standard analyses or the special analyses can be selected.

Standard analyses

The standard analyses consist of the analyses described for the INFORMATION operand of the START-NDM-DIAGNOSIS command (see page 341), except for INFORMATION=*PARAMETERS(...).

Prerequisites:

- The dump file (or the active system as the dump object) must be open.
- The system symbol library ($TSOS.SYSSMB.DAMP) must be available and must include the elements NDM, NDMNUC, DRV and possibly SHC-OSD for the system version to be analyzed. Otherwise, they are assigned as follows:

  ```
  ADD-SYMBOLS LIBRARY = *STD / <file> (ELEMENT = NDM / NDMNUC / DRV /
  SHC-OSD (VERSION = <version>))
  ```

- The system PRODAMP library ($TSOS.SYSDMP.DAMP) must contain the NDMDAMP PRODAMP objects.

  The system PRODAMP library is not predefined in DAMP by default. Consequently, it must usually be assigned as follows:

  ```
  ASSIGN-PRODAMP-LIBRARIES OBJECT-LIBRARY = *PRODAMP-SYSTEM-LIBRARY
  ```

  or in abbreviated format with:

  ```
  A-P-L O=*P.
  ```

  Instead of the keyword *PRODAMP-SYSTEM-LIBRARY, it is also possible to specify the path name of the library containing the NDMDAMP PRODAMP objects.
- The logical system file SYSLST can, if desired, be assigned to a file.

The procedures can be started by entering:

```
START-PRODAMP-PROGRAM NAME = NDM / NDMTRACE / NDMALL / NDMSTD / DRV / DRVALL
```

on the DAMP command line.

The PRODAMP procedures correspond to the analytical scope that can be selected with the INFORMATION operand of the /START-NDM-DIAGNOSIS command. The scope of the analysis for NDM is equivalent to that of NDMSTD, except for the fact that a restart is possible for NDM, but not for NDMSTD.

With the exception of NDMSTD, the analysis can be restarted in the event of errors with:

```
START-PRODAMP-PROGRAM NAME = NDM-RESTART
```

Note, however, that EDT must be available for the restart, since all the data relevant for the restart is contained in work file 9. This file must not be modified or deleted before the Restart call.

Special analyses

The special analyses are the analyses that can be individually set with INFORMATION=*PAR in the START-NDM-DIAGNOSIS command:

Prerequisites:

- The dump file (or the active system as the dump object) must be open.
- The system symbol library must be available and must include the elements NDM, NDMNUC, DRV and possibly SHC-OSD for the system version to be analyzed. Otherwise, an appropriate library must be specified when modifying the source NDM see below).
- The system PRODAMP library ($TSOS.SYSDMP.DAMP) must contain the NDMDAMP PRODAMP objects as well as the source NDM. Otherwise, they are assigned as follows:

  ```
  ASSIGN-PRODAMP-LIBRARIES SOURCE-LIBRARY = <prodamp-library>,
  ASSIGN-PRODAMP-LIBRARIES OBJECT-LIBRARY = <prodamp-library>
  ```

- The logical system file SYSLST can, if desired, be assigned to a file.

The settings for the options must be explicitly entered in the PRODAMP procedure NDM as described below:

- Loading PRODAMP dynamically:

  ```
  START-PRODAMP-EDITOR [abbr. PROC] <window-no> (between 4 and 9 or 21 and
  99)
  ```

- Reading the source NDM:
  - Overwrite “(procedure/index-identifier)” with “NDM”
  - Set MODE (field after “W<window-no>”) from “Dsp” to “Rea”
  - Send the screen with DUE (SEND)
- Modifying the source NDM:

  The source NDM essentially consists of the call to the PRODAMP procedure NDM_MAIN with appropriate parameters for the symbol library and the individual elements.
  The desired values from the specified value range are entered in the PRODAMP window. Note that the lengths of the individual parameters, which are predefined by the positions of the single quotes, must not be modified in the process.
  The significance of the individual parameters and the possible values are explained in the source on the following pages. The “+” and “-” keys can be used to scroll within the PRODAMP window.
- Compiling and executing the source NDM:
  - Set MODE=Go
  - Send the screen with DUE (SEND)

  The source with the entered parameters is compiled and executed.

6.1.2 Call from predefined ENTER jobs

The LMS library SYSENT.NDMDAMP.<ver> is supplied with the following ENTER jobs, which run under the TSOS user ID:

ENTER job	Analysis	File name	Restart possible
STD	Normal EDT and job variables are not used	SYSLST.NDMDAMP	-
NDM	Normal Job variables ($SYSJV.DATUM, $SYSJV.TIME) are used to generate the date and time.	SYSLST.NDMDAMP.<date>.<time>	x
NDMALL	Maximum	SYSLST.NDMDAMP.<date>.<time>	x
NDMTRACE	Trace data from NDM and BAVOLMON	SYSLST.NDMDAMP.<date>.<time>	x
DRV	DRV and traces	SYSLST.NDMDAMP.<date>.<time>	x
DRVALL	All DRV-relevant sections	SYSLST.NDMDAMP.<date>.<time>	x

Table 16: Predefined ENTER jobs for NDMDAMP

Note that the editor EDT must be available for all ENTER jobs where a restart is possible.

The ENTER jobs do not make use of SDF-P. They require and expect the standard file names of DAMP, e.g. the symbol files in $TSOS.SYSSMB.DAMP. If the standard names were not assigned when installing DAMP, they must be adapted accordingly.

Instead of the ENTER jobs, it is also possible to use the SYSSPR.NDMDAMP.<ver> procedure in batch mode, provided it is called with the ENTER-PROCEDURE command with at least the operand OBJECT=*SYSTEM specified.

6.2 Error handling during the analysis

Following an aborted analysis due to an error in NDMDAMP, it is possible to effect a restart at the next substep. The editor EDT must be available for the restart, since all the data required for the restart is stored in the EDT work file 9.

EDT is also required when enabling the PRODAMP trace to diagnose NDMDAMP errors, since the trace data for the NDMDAMP run is stored in the EDT work file 8.
This data is copied to the file NDMDAMP.TRACE at the end of the analysis if no restart has occurred, and the contents of the EDT work file 8 are then deleted.

In the case of a restart with the PRODAMP trace enabled, the data stored in the EDT work file 8 is transferred to the file NDMDAMP.TRACE.RESTART.<break#>. The substring <break#> specified in the file name designates the internal number of the PRODAMP procedure in which the analysis was aborted.
In order to enable an association between the various restarts and PRODAMP traces, a corresponding <break#> is indicated at the breakpoint even in the SYSLST output.
The trace data generated after the last restart and until the end of the analysis is stored in the file NDMDAMP.TRACE.

Since the trace files (NDMDAMP.TRACE, NDMDAMP.TRACE.RESTART.<break#>) can grow to be very large in size, errors may occur on saving the trace data if not enough storage space is available.
In such cases, NDMDAMP aborts the subsequent analysis.

6.3 Installation

NDMDAMP is installed with IMON.

If this installation is not performed correctly, NDMDAMP cannot be run properly. The most frequent errors and possible solutions for them are summarized below:

– DAMP reports that a PRODAMP object cannot be found (DMP4002).

  If the error message refers to the object NDM or NDM_RESTART, the PRODAMP library of NDMDAMP has not been merged into the general PRODAMP library ($TSOS.SYSDMP.DAMP).

  If the error message refers to NDEDAMP_CHECK_SYMBOLS, the PRODAMP library of SHC-OSD (SYSDMP.SHC-OSD.<version>) has not been merged. This library is called by NDMDAMP when an analysis of SHC-OSD is required and SHC-OSD is also loaded.

– NDMDAMP issues the message “Module NKATSOSM not found” and terminates.

  This means that either the dump object is not supported or the symbol library for BS2000 has not been correctly loaded.

– NDMDAMP reports that “no symbol file can be assigned” and terminates.

  This means that the symbol elements for NDM (NDMNUC, DRV and SHC-OSD, depending on the selected scope) have not been merged into the general or explicitly specified symbol library.

6.3.1 Release items for NDMDAMP

LOGICAL-ID	Description	Default name
SYSSDF.USER	Syntax file with the command START-NDM-DIAGNOSIS	$.SYSSDF.NDMDAMP.<ver>.USER
SYSSPR	S procedure	$.SYSSPR.NDMDAMP.<ver>
SYSENT	ENTER job	$.SYSENT.NDMDAMP.<ver>
SYSDMP	PRODAMP procedures, which must be incorporated into the system PRODAMP library of DAMP	$.SYSDMP.NDMDAMP.<ver>
SYSSII	Contains a description of the release items	$.SYSSSI.NDMDAMP.<ver>

Table 17: Release items for NDMDAMP

6.3.2 Logical units used by NDMDAMP

NDMDAMP does not evaluate the Logical Units of DAMP, but uses the default names set by DAMP:

– DAMP startup program

  For DAMP=*STD, the START-DAMP statement is used.

– Symbol library

  When SYMBOLS=*STD is set, the system symbol library of DAMP is used ($TSOS.SYSSMB.DAMP).

– PRODAMP library

  When PRODAMP=*STD is set, the system PRODAMP library of DAMP is used ($TSOS.SYSDMP.DAMP).

7 ELFE
Edit and evaluate the SERSLOG file

The utility routine ELFE edits the contents of a SERSLOG file or of all the SERSLOG files belonging to a given session.

A SERSLOG file is made up of individual records written by SERSLOG (see chapter “SERSLOG Software error logging in the SERSLOG file” on page 363) whenever an error event occurs.

ELFE provides an overview of the logged error events by drawing up a list of all the error event types in the file and the number of times each one occurs. ELFE can also be used to output information of specific attributes of individual error events (e.g. error event type, TSN responsible, time at which the error event occurred, etc.). The appropriate information can be output on the screen or (via SYSLST) on a printer in either complete or abridged form.

Each error entry is accompanied by a general description explaining the corresponding error event type (“rectype”) and offering diagnostic suggestions (which can be output either (via SYSLST) on a printer or on the screen).

ELFE processes all SERSLOG files, no matter which version of the operating system was used to create them. The SERSLOG file must not be active during processing.

7.1 Software and hardware prerequisites

The library containing the descriptions and diagnostic suggestions for SERSLOG files has the default name:

SYSLNK.ELFE.180	for SERSLOG files of BS2000/OSD-BC V9.0
SYSLNK.ELFE.190	for SERSLOG files of BS2000 OSD/BC V10.0
SYSLNK.ELFE.200	for SERSLOG files of BS2000 OSD/BC V11.0
SYSLNK.ELFE.200	Library contains the LLM.ELFE

It is not possible to evaluate two libraries in the same ELFE run. Assignment of a second library is rejected with error message ELF0012.

Supported terminal types and listing formats

ELFE supports all currently available terminal types.

The last line of the screen is reserved for inputs to the program.

The data is output to SYSLST with a format of 66 lines per page and 132 columns per line.

Storing SERSLOG files

SERSLOG files are generated using the logical block size STD(1).

Using aliases

The user can use ACS (alias catalog service) to define aliases for SERSLOG files and for the description library for the record types. These aliases can be passed to ELFE.
If an alias is assigned to a SERSLOG file which does not exist, ELFE uses the alias in place of the real file name in order to document errors which occur when processing this file.

ELFE messages

The ELFE messages have the message class ELF. Information on individual messages can be obtained in ongoing operation with the /HELP-MSG-INFORMATION command.

7.2 Operation

The ELFE program is called by means of **/START-ELFE**.

`OPEN filename` is used to select the SERSLOG file and to open or select a specific session and open its files. You select an evaluation library using the `LIBRARY` statement. You can then examine the error event entries more closely. `STOP` or `END` terminates the program.

Statement	Function
C(ONT)	Process a system run whose auxiliary files are still available
D(ISPLAY)	Display information on the screen
E(ND)	Terminate ELFE
H(ELP)	Display help information on ELFE statements on the screen
K(EEP)	Retain auxiliary files at the end of the program or on changing the system run
L(IBRARY)	Assign an evaluation library
O(PEN)	Open SERSLOG files
P(RINT)	Print error entries
S(TOP)	Terminate ELFE

Table 18: Overview of ELFE statements

CONT
Continue evaluation of SERSLOG file or session

With the aid of the CONT statement, evaluation of a SERSLOG file or of a system run which has already been started under the same user ID can be continued, providing the auxiliary files which were created in the previous session are still available (see the KEEP statement, page 359). If another SERSLOG file or another session is currently being evaluated, CONT terminates this evaluation.

Format

Operation	Operands
C(ONT)	sss

Operands

sss Three-digit number of the session.

DISPLAY
Display error entries on screen

The DISPLAY statement is used to display information about the system or error entries from the SERSLOG file on the screen. The error entries can be selected by specifying the appropriate operands for a number of different criteria, e.g. the error event type, the TSN or the time of the event. The selected entries can be displayed either in their entirety, together with the error environment data, or in an abbreviated form with only the most important details.

Format

Operation	Operands
D(ISPLAY)	INFO LOCMAP FULL ADDRESS=address MODULE=modulename DESCRIPT[,RECTYPE=rectype] SHORT SUMMARY ELSN={elsn / elsn1-elsn2} [TSN=tsn] [TID=tid] [,...] [,SHORT] TIMESTP=date1/time1:date2/time2 RECTYPE=rectype

Operands

ADDRESS=address
Name of the module within which the specified address is located, together with the displacement from the start of this module. “address” must be specified as a 4-byte hexadecimal value.

DESCRIPT
This operand causes descriptions of the specified error event types (“rectypes”), accompanied by suggested diagnostic responses, to be displayed. DESCRIPT should be specified together with the RECTYPE operand, otherwise all the descriptions contained in the description library will be output. The DESCRIPT operand is accepted only if a description library has been assigned (see the LIBRARY statement, page 359).

ELSN=$\left\{\begin{array}{l}\text{elsn}\\ \text{elsn1-elsn2}\end{array}\right\}$
This selects the entries on the basis of their ELSN (Error Log Sequence Number); the entries are numbered sequentially during creation of the SERSLOG file. “elsn”, “elsn1” and “elsn2” must be specified in hexadecimal form. “elsn2” must be greater than “elsn1”. The specification “elsn1-elsn2” selects all entries whose ELSN lies between these limits (including the specified values).

FULL
This specifies that all information on the system, the location map of the modules, all entries (in ascending order of their ELSN) and the SUMMARY list are to be output.

INFO
This outputs information on the system (version, generation date, etc.).

LOCMAP
This outputs the location map of the system modules. The map contains two lists, one sorted by module names and one sorted by module addresses.

MODULE=modulename
This outputs the start address, the length and the version number of the specified module.

RECTYPE=rectype
The entries to be output are selected on the basis of the error event type (“rectype”). The entries are output in ascending order of their ELSN. If less than 7 characters are specified, all entries whose error event type begins with the specified string are output (r[e[c[t[y[p[e]]]]]]).

SHORT
This outputs the selected SERSLOG entries in their abbreviated form. The SHORT operand may be specified together with any of the selection operands.
If SHORT is the only operand specified, all entries in the SERSLOG file are output in their abbreviated form.

SUMMARY
: This outputs a list containing all error event types found in the current file or the current session, together with the number of times each type occurs.

TID=tid
: The SERSLOG entries to be output are selected on the basis of the task identifier. The entries are output in ascending order of their ELSN.

TIMESTP= date1/time1:date2/time2
: The entries to be output are to be selected using the ELSN written to the SERSLOG file in the period date1/time1 through date2/time2.

  Output is in ascending sequence of ELSN. The date and time are specified in SDF format. The upper limit specified by date2/time2 must be greater than or equal to the lower limit specified by date1/time1.

  Example

```
DISPLAY TIMESTP=2008-10-25/12:45:00:2008-10-25/14:00:00
```

TSN=tsn
: The entries to be output are selected on the basis of the TSN. The entries are output in ascending order of their ELSN. If a non-numeric TSN with less than 4 characters is specified, the entry is padded on the left with blanks.

Example

The statement

```
DISPLAY TSN=1234,RECTYPE=NRT,SHORT
```

causes all SERSLOG entries with the TSN 1234 whose RECTYPE begins with the string “NRT” to be output in their abbreviated (short) form. The entries are output to SYSOUT in ascending order of their ELSN.

If the number of entries to be output is greater than the number of lines on the screen, one screen is displayed and the message

```
ENTER '+' OR NEW COMMAND
```

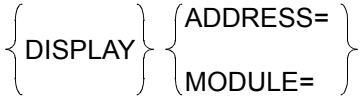
is displayed at the bottom of the screen. Entering “+” or a null input (simply pressing DUE) causes the next screen to be displayed. Entering a new command terminates the output and the new command is executed.

The following commands have the following effects:

STOP/END:	Abort the current statement, processing of the session or file and the ELFE session.
OPEN/CONT:	Abort the current statement, processing of the session. Start with the specified session.
PRINT/DISPLAY:	Abort the current statement and begin processing the specified statement.

Exceptions

The only exceptions to this are the statement



and

the statements KEEP, HELP and LIBRARY
These interrupt processing of the current information output only for output of the requested data. The previous information output can then be continued.

All other operands of the DISPLAY statement terminate any current information output.

END
Terminate ELFE

The END statement terminates the ELFE utility routine and aborts any processing that is still in progress. It also deletes the auxiliary files created during the session unless this has been precluded by means of the KEEP statement.

Format

Operation	Operands
E(ND)	

HELP
Display brief information on ELFE statements

The HELP statement enables the user to request help information on any of the ELFE statements. If HELP is entered without an operand, information on all statements is displayed.

Format

Operation	Operands
H(ELP)	[statement]

Operands

statement The statement on which information is to be displayed.

KEEP
Retain auxiliary files

ELFE works with the auxiliary files S.SERSLOG.sss.ELSN, S.SERSLOG.sss.INFO, S.SERSLOG.sss.MODULE and S.SERSLOG.sss.ADDRESS. These auxiliary files are normally deleted when the ELFE program is terminated. However, the KEEP statement can be used to retain them after termination, e.g. if it is necessary to interrupt processing.

Exception

If no operand or a file name was specified in the OPEN statement, the KEEP statement is rejected.

Format

Operation	Operands
K(EEP)	

LIBRARY
Assign description library

The LIBRARY statement can be used to assign a library which contains descriptions of the error event types and diagnostic responses.

Format

Operation	Operands
L(IBRARY)	filename

Operands

filename — Name of the library to be assigned; see page 352. It is not possible to assign more than one library during an ELFE session.
Any attempt to assign a second library will be rejected with error message `ELF0012`.

OPEN
Assign and open file to be evaluated

The OPEN statement is used to specify and open the files to be evaluated. The necessary auxiliary files are created at the same time.
OPEN may be specified with a complete file name, with the three-digit number of a specific session (together, if applicable, with the sequence number of the file), or without operands. ELFE then locates and opens the appropriate file. If, necessary, this file must be stored under the user's own user ID.
If another file or session is currently being processed, OPEN terminates this processing and initiates processing of the specified file or session.

Format

Operation	Operanden
O(PEN)	[sss[,nn][,STD-NAME=NEW/OLD] | filename]

Operands

sss — Three-digit decimal number of the session.
All files with the standard name "SYS.SERSLOG.yyyy.mm.dd.sss.nn" which exist under the caller's user ID are included for evaluation, where "sss" is the specified session number and "yyyy.mm.dd" and "nn" may have any value. The date specification in the standard name uses the new format by default (see the STD-NAME operand).

nn — Two-digit decimal consecutive number of a SERSLOG file within a session.
If both "sss" and "nn" are specified, the file under the caller's user ID with the standard name SYS.SERSLOG.yyyy.mm.dd.sss.nn is evaluated, where "sss" is the specified session number, "nn" is the specified consecutive number of the file within the specified session, and "yyyy.mm.dd" may have any value. The date specification in the standard name uses the new format by default (see the STD-NAME operand).

STD-NAME=

NEW	The date in the file name is specified as yyyy-mm-dd.
OLD	The date in the file name is specified as yy.mm.dd.
filename	Name of the SERSLOG file to be evaluated.

If, before calling the ELFE utility routine, the user enters a file with the link name SERSLOG in the task file table (TFT) by means of an ADD-FILE-LINK command, the operands of the OPEN statement may be omitted. ELFE then opens the file linked with the link name SERSLOG.

Continuation of the screen output

If the amount of data to be output exceeds the number of screen lines, the contents of one screen are output and the user then controls whether the current option is continued or canceled.
Entering "+" or null input (DUE only) continues output.

Input options and effects:

S/END:	Cancels the current statement, processing of the session or the file and the ELFE run.
O/C:	Cancels the current statement and processing of the session. Starts with the specified session.
P/D:	Cancels the current statement and starts processing the specified statement. If a command was entered and accepted during the process of opening a session, the session is considered open. Evaluation is performed on the files open up to that point.
H/L	Interrupts the current statement relating to processing this request. The original statement is then edited further.
+	Continues the current statement.

Parallel calls

If ELFE is called in parallel, the DMS error 05B1 can occur as a result of auxiliary files having the same names. The problem can be circumvented by specifying the number of the session in the OPEN statement (OPEN 003,STD-NAME=NEW).

PRINT
Print error entries

The PRINT statement allows you to output information on the system or error entries from the SERSLOG file (via SYSLST) to a printer. By means of specifying the appropriate operands, the information can be selected on the basis of various criteria, such as the error event type, the TSN or the time of the event. The selected entries can be printed either in their entirety, together with the error environment data, or in an abbreviated form with only the most important details.

Format

Operation	Operands
P(RINT)	{ INFO LOCMAP FULL ADDRESS=address MODULE=modulename DESCRIPT[,RECTYPE=rectype] SHORT SUMMARY { ELSN={ elsn / elsn1-elsn2 } [TSN=tsn] [TID=tid] } [,...] [,SHORT] TIMESTP=date1/time1:date2/time2 RECTYPE=rectype }

The operands of the PRINT statement are the same as those of the DISPLAY statement (see page 354).

STOP
Terminate ELFE

The STOP statement terminates the ELFE utility routine and aborts any processing that is still in progress. It also deletes the auxiliary files created during the session unless this has been precluded by means of the KEEP statement.

Format

Operation	Operands
S(TOP)	

8 SERSLOG
Software error logging in the SERSLOG file

Software error logging in BS2000 consists of two parts: saving the data on all software errors encountered, and editing this data. Data relevant to the software errors encountered is saved with the aid of the operating system function SERSLOG. Selected data relating to each software error is written into a special file, the SERSLOG file. In order to avoid impairing system performance, this data is not subjected to further editing at this point. It can be edited and evaluated later with the aid of the utility routine ELFE (Error Log File Evaluation, see page 351).

The following overview contains all the commands available to the operator and the system administrator for controlling software error logging. The commands are described in detail in the “Commands” manual [8].

Commands	Function
CHANGE-SERSLOG-FILE	Closes the current SERSLOG file and opens a new one.
SHOW-SERSLOG-STATUS	Shows the status of error logging and the name of the SERSLOG file.
START-SERSLOG	Starts software error logging and opens a SERSLOG file.
STOP-SERSLOG	Closes the SERSLOG file.

Table 19: Overview of SERSLOG commands

SERSLOG file

A SERSLOG file is made up of individual records written by SERLSOG when an error occurs. Each record comprises the ELSN (Error Log Sequence Number), the designation of the error event type (“rectype”), the TSN, the TID, the name of the module which caused the entry, the time of the error event and data from the environment of the software error.

The SERSLOG file is opened during system startup when software error logging is activated. The name of the SERSLOG file has the following format:

`SYS.SERSLOG.yyyy-mm-dd.xxx.nn` bzw. `SYS.SERSLOG.yy.mm.dd.xxx.nn` (depending on the setting for the system parameter FMTYFNLH – for more details see the manual “Introduction to System Administration” [6]).

where:

`yyyy-mm-dd`	is the date on which the file is opened.
`xxx`	is the number of the associated session.
`nn`	is the sequence number of the SERSLOG file (01 to 99, always begins with 01 at startup time). If nn becomes greater than 99, the counter is reset to 01; this causes the first SERSLOG file to be overwritten.

The SERSLOG file is not write-protected.

When the system run is terminated, the SERSLOG file is closed and software error logging is terminated. The current SERSLOG file is included in the SLED output.

Only the operator and the system administrator can activate (START-SERSLOG) or deactivate (STOP-SERSLOG) software error logging or switch the SERSLOG file (CHANGE-SERSLOG-FILE). The SHOW-SERSLOG-STATUS command allows you to request information on software error logging.

9 ASE
Auxiliary SERSLOG Extensions

The ASE (Auxiliary SERSLOG Extensions) subsystem permits automatic monitoring of critical system statuses, which are reflected in SERSLOG events. Threshold values can be defined for these events which, when they are exceeded, result in the events being logged in one of the following ways: in an internal buffer, by a message at the console, and/or via Remote Service. This logging can be restricted to selected SERSLOG events.

The following overview contains all the commands available to the operator and the system administrator for controlling software error logging. The commands are described in detail in the “Commands” manual [8].

Command	Function
ADD-ASE-ELEMENT	Declares an ASE element
MODIFY-ASE-PARAMETERS	Changes global ASE settings
REMOVE-ASE-ELEMENT	Deletes ASE elements
SHOW-ASE-ELEMENT	Displays ASE elements
SHOW-ASE-LOGGING	Displays internal ASE logging data
SHOW-ASE-PARAMETERS	Displays global ASE settings
SHOW-ASE-STATUS	Provides ASE status information

Table 20: Overview of the ASE commands

10 SLED dump

If it is not possible to pinpoint the cause and effect of a software error which impairs an essential part of the operating system, the operating system must be terminated and the memory areas of BS2000 must be dumped for diagnosis. A full dump of this type is created by SLED (Self-Loading Emergency Dump routine).

SLED runs independently of the BS2000 operating system.
The operating system has to be loaded again following a SLED run.

SLED execution can be either automatic (unattended) or controlled by the operator (attended).

The SLED run can be controlled via a SLED parameter file.

SLED writes a dump file (SLEDFILE) to disk or tape. This file contains all the available, requested data that is necessary for subsequent analysis by the editing routine DAMP.

If SLED was loaded as “DUMP from SLED”, the memory areas used by IPL-EXEC and SLED are output.

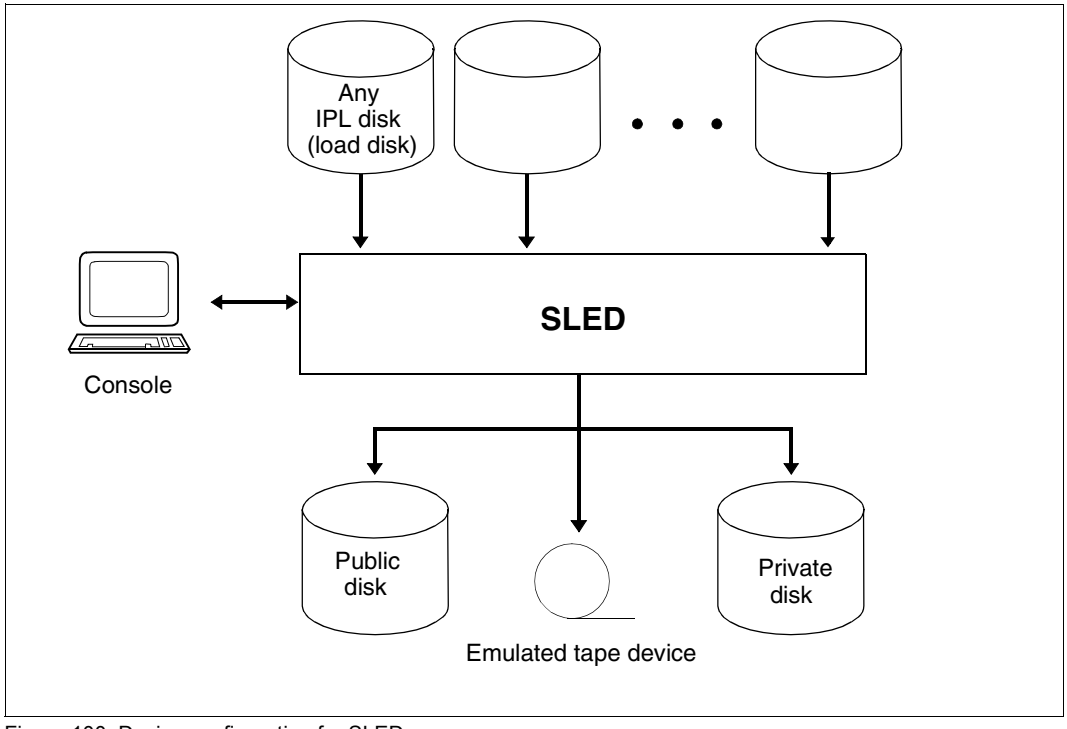


Figure 100: Device configuration for SLED

10.1 Loading and initializing SLED

SLED runs as an application program under IPL-EXEC. IPL-EXEC is part of the initial program loader (IPL), which is loaded and initialized before SLED is loaded.

When loading, a distinction must be made between loading SLED for the first time after a system crash, repeating a SLED run and dumping from the SLED system.

Before loading SLED for the first time the memory areas used by IPL, BOOT and SLED must be saved. This is done partly by the firmware (by copying data areas to save areas in memory or to the service processor) but for the most part by the software by writing data areas to the IPL disk to a save file (SLEDSAVE or BOOTSAVE) before they are used by BOOT, IPL or SLED.

Initialization of SLED varies according to server type. It is described in full in the appropriate manuals for each server type.

For the loading procedure, the load disk is searched for the following files, which must be anchored in the SVL of the disk with SIR:

– $TSOS.SYSPRG.BOOT.DSKnnn.SAVE (BOOTSAVE)
– $TSOS.SYSPRG.IPL.DSKnnn (IPL phase)
– $TSOS.SYSREP.IPL.DSKnnn (corrections for IPL)
– $TSOS.SYSREP.SLED.DSKnnn (corrections for SLED)
– $TSOS.SYSPRG.SLED.DSKnnn.SAVE (SLEDSAVE)

Before SLED is loaded, all the disks required for subsequent processing should already have been mounted and switched online.

Once SLED has been loaded and started, a number of consistency checks are carried out to establish:

- whether the version of the loaded SLED matches that of the loading IPL
- which system was previously loaded and if this was BS2000 (also under VM2000 or for DUMP from SLED) whether its version matches that of SLED
- whether a part of main memory was overwritten without having been saved.

SLED performs these consistency checks regardless of whether the dump runs attended or unattended. Unattended operation means that SLED runs automatically without operator intervention, using the presettings in the SLED parameter file or default settings. Attended operation means that SLED prompts the operator to enter or correct options and SLED is controlled interactively.

Appropriate warnings regarding execution of the consistency checks are output at the console and logged in the SLED output file.

Diagnostic data can be output to the public disks, to private disks or tape.

The appropriate response must be entered to the following message during an attended SLED run:
`NSD1003 STANDARD SLED ? REPLY (Y; N; EOT=Y)`

Standard SLED

By entering `Y` or `<EOT>` in response to message NSD1003, the operator selects standard SLED. This results in the standard SLED behavior described below:

1. In both attended and unattended operation, a check is made as to the availability of the disks of the home pubset and the paging disks (online scan). Unavailable (offline) disks are logged via message `NSD1400`. If data from the missing disks is subsequently required, SLED must be reloaded once these disks have been attached; it is not possible to attach required disks during the SLED run.

   The more pubsets SLED has to include (the home pubset of the system to be dumped, the load pubset of SLED and possibly another pubset for the parameter file and one for the SLED output file), the longer the online scan takes.

2. The default values for the MODE and TASK parameters (described below) are initialized as EOT. Regardless of whether or not there has been a preceding system crash, SLED selects the settings MODE=STD (see page 375) and TASK=STD (see page 376).

   Once the SLED initialization phase is complete, it becomes known whether
   – all disks of the home pubset of the aborted session are online
   – public disks from different pubsets are online
   – all paging disks used in the aborted session are online

The operator dialog in standard SLED is continued with message `NSD5200` (assignment of a parameter file).

Nonstandard SLED

By entering N in response to message NSD1003, the operator selects nonstandard SLED. The operator requests an extended dialog with SLED for controlling execution and receives the following additional messages:

```
NSD0900 ONLINE SCAN ? REPLY (Y; N; IPL-CONF=I; GENERAL ONLINE SCAN=X; EOT=Y)
```

SLED asks whether an online scan is to be performed, i.e. whether the available device configuration is to be checked.

Y IPL performs an online scan for each pubset required.
The behavior is the same as with standard SLED.

N IPL is not to perform an online scan
In this case only the SLED loading disk is available. The operator should only select this value if SLED is not to be executable with an online scan or IPL-CONF evaluation.

I Instead of an online scan, the system-specific partition in the file containing the current configuration for system initialization, $TSOS.SYSDAT.IPL-CONF.<ver>, is evaluated. If this partition does not exist in the file or if errors occur during processing, an online scan is initiated again for the pubsets required. After successful processing, SLED knows the home pubset, the paging disks and the SLED load disk if the associated disks were already attached when the system was booted and could therefore be entered in the IPL-CONF file.

X IPL performs an online scan for all disks.

SLED cannot use the general online scan for large systems, since it cannot manage more than 1290 disk device entries. This option should only be selected in special cases

The response is followed by two further messages which the operator must answer. The responses determine the scope of the SLED file.

```
NSD3001 SPECIFY NOEDIT MODE.
        REPLY (STD; NSF; REAL; ALL; EOT=STD; - (BACKTRACK))
```

For a description of this message, see page 375.

```
NSD3002 SELECT TASKS.
        REPLY (STD; NONE; ALL;(TSN LIST); EOT=STD/ALL; - (BACKTRACK))
```

For a description of this message, see page 376.

Error conditions for standard and non-standard SLED

Although SLED can run if the SYSRES (system disk) is not available, no system files (TSOSCAT, logging files etc.) and possibly no paging area data can be saved.

If the home pubset is only partially available, some system files may be only partially saved.

Unavailable paging disks may result in incomplete diagnostic information.

If SLED determines that one of the required disks is missing, this is logged via message `NSD1400`. The operator can attach the missing disks and repeat the SLED run.

Repeat SLED

A repeat SLED refers to the loading and initialization of another SLED run following a first SLED run in order to obtain the dump of the previously aborted session. This may be necessary, for example, if a SLED was inadvertently loaded which was not compatible with the version of the aborted system or the disks required during SLED initialization were not available (online).

Consequently, with a repeat SLED the areas saved in BOOTSAVE and SLEDSAVE must be used again and must not be saved a second time.

Both firmware and software attempt to recognize a repeat SLED and in this case suppress the saving of data areas. This means that no data is lost in the case of a repeat dump. If the newly loaded SLED does not belong to the same operating system version as the first SLED, it may be that the repeat SLED is not recognized and some of the diagnostic data is lost.

SLED dump

If an error occurs during a SLED run (message `NSD1002`), it may be necessary to take a dump from SLED. This means that SLED is loaded again in order to generate information on the errored SLED run.

Therefore, even though SLED was already loaded, it is necessary in this case to save the memory areas in which SLED was loaded a second time since these areas are required for SLED diagnostics. Both in the firmware and in the software it is necessary to note the following:

- In the event of a SLED repetition, the data overwritten by BOOT, IPL and SLED has already been saved and is therefore **not** saved again.
- In the case of a SLED dump, the data overwritten by BOOT, IPL and SLED (SLED data) is **now** saved again.

It is therefore necessary to take special measures if it is necessary to generate a dump relating to the execution of the dump function itself (SLED dump):

- In VM2000 operation, this is achieved at VM start (`/START-VM`) by specifying the parameter `UNLOCK-SAVEAREA=*YES`
- On x86 servers at system start with `ipl parameter [d|u]: u` (UNLOCK).
- However, on /390 servers it is necessary to perform certain actions. The selection of the actions depends on the server in question. For a detailed description see the manuals for the various server types.

  The actions may include, for example:

  – Stop CPU
  – Log register contents
  – Set the real address stop X'4000'
  – On multiprocessor systems, set START/STOP mode to TARGET CPU
  – Start dump function
  – After the address stop is effective, overwrite real memory location X'1800' with X'00'
  – Reset address stop
  – Start CPU
  – Proceed as for dump function
  – Reset START/STOP mode after termination of dump function

Function selection

The output medium is selected by responding to the following message:

```
NSD3000 SPECIFY OUTPUT DEVICE.
        REPLY (DPUB; DPRIV; TAPE; PRINTER; EOT=DPUB; -(BACKTRACK))
```

Possible responses:

`DPUB`	output to public disk (default value)
`DPRIV`	output to private disk
`T[APE]`	output to tape
`P[RINTER]`	output to printer (obsolete)

10.2 Output to a dump file

SLED generates a dump file (SLEDFILE) which can be prepared and analyzed by the DAMP dump analysis routine.

Defining the output data

The scope of the output data to be written to SLEDFILE is defined by means of the parameters MODE (as a response to message `NSD3001`) and TASK (as a response to message `NSD3002`). The MODE parameter determines the selection of memory pages to be included in the SLEDFILE. The TASK parameter determines the tasks whose address space is to be saved.

The parameters MODE=ALL and TASK=ALL are set automatically when:

- the main memory is less than 128 MB
- the system tables for the page selection are corrupt
- the product ID or dump testament contains an error.

In the case of a standard SLED (i.e. automatic SLED or the response to `NSD1003` is `EOT` or `Y`), the MODE and TASK parameters can be specified only by being entered in advance or via the parameter file. If the two parameters are not specified, SLED itself defines the values (implicit EOT response).
You are recommended not to specify the MODE and TASK parameters and instead allow SLED to define these values.

Page selection using the MODE parameter

```
NSD3001 SPECIFY NOEDIT MODE.
        REPLY (STD; NSF; REAL; ALL; EOT=STD; – (BACKTRACK))
```

Depending on the response, the following pages from main memory and the paging area are output:

`EOT` (no input)
: The value of this parameter is determined by SLED:
SLED selects `MODE=STD` independently of whether or not a system crash previously occurred (SETS). In the case of a system crash, the task or module that caused it is usually included in the scope of the output.

`STD`
: The following pages are output:
– pages of the privileged data spaces
– class 1 through class 4 memory pages (system address space)
– class 5 pages of all selected tasks
– class 6 pages of all selected system and SVC79 tasks
– class 6 pages of all tasks specified in the TSN list
– resident*) class 6 pages of all TICs (task in control)
– resident*) class 6 page 0 of all selected tasks.

*) “resident” in this context means that the page is located in main memory.

`NSF` (No System Files)
: Has the same effect as `STD`, but without the system files that are also saved if `STD` is specified, provided that they are accessible.

`REAL`
: All main memory pages (the subsequent TASK parameter is ignored); no data in the paging area is saved.

`ALL`
: In addition to the pages selected by `MODE=STD`, the entire main memory is output.

If `MODE=ALL` then the SLED dump can become extremely large!

Task selection using the TASK parameter

In addition to the system address space (classes 1 through 4), the address space of the specified tasks can also be saved, depending on the value of the TASK parameter.

```
NSD3002 SELECT TASKS.
        REPLY (STD; NONE; ALL; (TSN LIST); EOT=STD/ALL; -(BACKTRACK))
```

`EOT` (no input)
: The value of this parameter is determined by SLED:
SLED selects `MODE=STD` independently of whether or not a system crash previously occurred (SETS).

`STD`
: The output includes:
– all TICs (tasks in control)
– all system tasks
– all privileged (SVC-79) tasks
– all CDUMP in progress tasks and dump tasks
– all tasks in queue Q10 (Permanently Pended)
– all tasks from the TIC trace tables (i.e. the last 64 tasks assigned to a logical machine

`NONE`
: All TICs (tasks in control) on a CPU.

`ALL`
: All tasks

`<tsn1>,<tsn2>,...,<tsn8>`
: In addition to the tasks listed under `STD`, the tasks specified in this list (maximum 8) are saved in the dump.

SLEDFILE contents (MODE = STD/ALL)

1. STATUS section (CPU status)
2. MAINMEM section: selected main memory pages
3. HSA section (only for /390 servers operated in native mode)
4. VM2HYPVS section (VM2000 Hypervisor on /390 servers, if a SLED has been created in a VM2000 guest system)
5. IOHIOSDP (bus dump file; only on x86 servers)
6. FIRMWARE section: firmware code und datea (only on x86 servers)
7. PAGEPHYS section: selected pages of the paging area
8. PROTKEYS: memory protection key

The following may also be present if the data is accessible and BS2000, IPL, SYSTART, VM2000 or SLED has been loaded:

9. TSOSCAT: system catalog
10. EQUISAMQ: SPOOL job queue
11. SJOBPOOL: job management queue
12. REPLOG: if it is not possible to access the REPLOG file then SLED saves the SAVEREP file which contains only the BS2000 repairs.
13. CONSLOG: last console logging file of this session
14. CONSLOG1: first console logging file of this session
15. CONSLOG2: penultimate console logging file of this session
16. HELFILE: hardware error logging file - HEL
17. SERSLOG: last software error logging file of this session
18. SERSLOG1: first software error logging file of this session
19. SERSLOG2: penultimate software error logging file of this session
20. MSCFTRAC: MSCF trace file
21. SJMSFILE: JMS file
22. PAGELOG section: table of tasks saved on last system abort
23. SLEDMEM section: IPL and SLED coding of the current SLED run
24. SLEDLOG section: recording of start of SLED dialog

If `REAL` is specified then the SLEDFILE contains the items 1, 2, 3, 4, 5, 6, 7, 9, 24 and 25.
If `NSF` is specified then the SLEDFILE contains items 10 to 22.

The operator can initiate a maximum dump by specifying MODE=ALL and TASK=ALL as a response to message `NSD3001` or `NSD3002`.

Output to an emulated tape device

On all BS2000 servers two tape devices are configured which are emulated by the Management Unit (SU /390), the SKP (S servers) or X2000 (SU x86). One of the tape devices operates in real mode on the basis of the integrated CD/DVD drive. The other operates on the basis of a file which is stored in the file system of the MU, SKP or X2000. In addition, further tape devices operating on file basis can be configured.

An emulated tape device operating on file basis must be used for SLED output. Tape devices which operate on the basis of a CD/DVD drive cannot be used for SLED output.

SLED output to an emulated tape device is provided for situations in which a SLED file on disk is not available. SLED output is in particular not convenient in large system configurations in which continuation tapes are required and calls for a certain degree of preparation.

The tape in the tape device must already have been initialized, i.e. assigned standard labels (VOL1, HDR1 and HDR2). Neither the volume serial number (VSN) nor the recording density can be changed. A check is also carried out to determine whether the expiration date entered in label HDR1 has been reached.

In older servers and firmware versions the tape visible in the preconfigured tape device will not yet have been initialized.
If the missing initialization is not to be implemented later (e.g. using the INIT utility routine), subsequent SLED output to the tape will not be possible.

The SLED output file to tape is always named SLEDFILE.

SLED requests two entries via messages `NSD3800` and `NSD3822`: the volume serial number (VSN) and the device identifier (device mnemonic).

1. Volume serial number (VSN)

   The VSN may be specified as a fully or partially qualified entry. The asterisk (`*`) is used as a wildcard symbol (only allowed at the end of the entry). If `*` is entered by itself as the VSN, SLED will accept all tapes, provided they have standard labels.

2. Device identifier

   The device mnemonic `mn` is specified as the device identifier. SLED checks whether the specified device exists and whether it can be used for output.
   If the data specified is invalid, SLED repeats its request for the device identifier.

   These messages likewise appear if **one** tape is not sufficient to accommodate the entire output and a continuation tape has to be used. If the VSN was specified as `*` this also applies to the subsequent tapes.

   All continuation tapes must be mounted on the same device.

Because of the emulation, a continuation tape is “mounted” only by means of the following actions:

1. Backup of the file belonging to the tape emulation, e.g. by downloading it to a PC.
2. Overwriting the file belonging to the tape emulation with a prepared file which represents an empty tape with a different VSN. This is done, for instance, by uploading from a PC.

   The procedure for the two steps is described in the operating instructions of the server concerned.

For reasons of data security, the files written should be overwritten (e.g. by reinitializing the tape) or physically deleted after they have been analyzed.

Output to private disk

In the case of output to private disk, the SLED output file (SLEDFILE) must be contained completely on one disk, i.e. it must not be distributed over several disks. The file must already have been created and must be sufficiently large.

The operator is requested to specify the VSN of the disk. The device address of the disk is then queried via message `NSD3410`.
An operator wishing to use only the device name can enter `*` or `<EOT>` in response to message `NSD3400`. The parameter file should contain: `VSN=*, DEV=<mn>`.

Once the output disk has been defined and located, the name of the output file is requested. If output is to private disk, the file is not accessed via the system catalog but exclusively via the F1 labels on the disk. Consequently specification of a catalog ID is irrelevant in this case and is rejected as an error.

If there is already a catalog entry for the file on private disk, it is not updated. After a SLED output file has been created on a private disk, the corresponding catalog entry must be deleted by means of the command EXPORT-FILE FILE-NAME=<filename>.

SLED files must not be created on DRV private disks.

Output to public disks

In the case of output to a public disk, the pubset containing the output file must first be identified. The output files for SLED can also be located outside the home pubset, but only on disks or pubsets which would be suitable as an IPL disk or home pubset, i.e. for example, not on SM pubsets. Output to shared pubsets is also rejected.

The pubset of the SLEDFILE is identified via the first file name to be requested. The following rules apply:

1. If the file name was specified with catalog ID, this suffices to specify the pubset containing SLEDFILE.
2. If the file name was specified without with catalog ID (or if the default name was specified implicitly by a null input), an attempt is made to determine the pubset using one of the following two standard rules:

   a) SLED was loaded from a public disk: the pubset to which this disk belongs is the pubset containing SLEDFILE.

   b) SLED was loaded from a private disk but all the public disks that are online belong to a single pubset: this is then the pubset containing SLEDFILE.

   If neither of these rules is applicable, this means that SLED was loaded from a private disk and that there are public disks online from various pubsets or no public disks online at all. In this case the operator is requested to specify the catalog ID of the SLED output file.

If the pubset containing SLEDFILE is known, first the associated SYSRES and then all the other disks of the pubset are sought. SLED cannot execute unless all the disks of the SLEDFILE pubset are online. If disks are missing, this is indicated by message `NSD1400`, SLED must be reloaded once these disks have been attached (SLED repetition!).

Subsequently an attempt is made to locate the specified output file. To this end an accessible catalog with the specified catalog ID must be available.

If the software product HSMS is used on the system involved, systems support must ensure that the file to be output is not automatically migrated and thus made inaccessible if it is not used for a long time.

A pubset for SLED output must not be imported by a running system during SLED operation.

Checking the SLED output file

Once the output file (SLEDFILE) has been identified and located, it must be checked to ensure that it is possible to work with it. This means:

1. It must not be protected by a password.
2. ACCESS=WRITE must have been specified.
3. The expiration date must have been reached.
4. It must be large enough to accommodate at least the main memory dump, the CONSLOG and SERSLOG files and the hardware data. SLED writes very large main memories as a number of different portions so that DAMP is able to prepare the dump even if the main memory could not be saved in full.
   However, excessively small SLED files should be avoided since precisely that data might be missing that is required for error diagnosis.
5. If output to public disks has been requested, the file must not have been created on private disk.
6. Output to shared pubsets and SM pubsets is prohibited and is rejected by IPL.

If any of these conditions is not satisfied, the appropriate message (`NSD32xx`) is issued and the name of the SLED output file is requested again.

If the SLED file is not logically empty, it is not used unless the operator makes a positive response to the appropriate query (message `NSD3204`); otherwise the name of the SLED file is requested again.

Size of SLED output files

If SLED output is to disk, a sufficiently large file must be available on disk. The output file may also be greater than 32 GB (but at most 256 GB) and may reside on the home pubset.

As SLED operates without DMS support, the file cannot be extended dynamically during a SLED run. A sufficiently large value must therefore be specified in the SPACE operand of the CREATE-FILE command when creating a SLED output file.

The scope of the dump, i.e. the size of the SLED output file, is influenced by many factors that are unknown before the dump is taken. At the time the dump is taken, the size of the output file can be controlled via the MODE parameter (pages to be included in the file) and the TASK parameter (tasks whose address space is to be saved). Setting the parameter TASK=ALL to include all tasks' address space has a significant effect not only on the time consumed by the SLED run but also the file size. TASK=ALL can result in a file many times the size of that produced with TASK=STD.
The MODE parameter is decisive if the main memory or system files are large. In such cases, MODE=ALL should only be specified if the file is sufficiently large.

The following factors must be taken into account when calculating the required file size for MODE=STD, TASK=STD (default case):

A: Size of the system address space used

B: Size of the system files TSOSCAT, EQUISAMQ, REPLOG, CONSLOG[x], SJOBPOOL, HEL, SERSLOG[x], SJMSFILE, MSCFTRAC, etc.

n: Number of CPUs

t: Number of different tasks entered as tasks in control in the TIC table (maximum 64 entries per processor)

C: Size of a task's class 5 address space used

The influence of these factors is calculated using the formula (A + B + n * t * C)

Factors t and C are particularly problematic when calculating the file size. Simply setting the upper limit for these factors results in impractically large values. It is difficult to calculate average values that are generally applicable, since these values vary widely according to the way in which the system is used and the system workload. It may be possible to use the average results of *open*SM2 measurements in this case.

To summarize, it is difficult to recommend a size for the output file. In most cases, however (except where TASK=ALL is specified), the user will find that double the size of main memory is sufficient.

Rule of thumb for MODE=STD/ALL, TASK=STD

Size of `SLEDFILE = 2 * size of main memory`.

Example for a SLED run

```
%S.NSI00E3  IPL-REPS READ: 0; EXECUTED: 0
%S.NSI1100  IPL DEVICE = HIP6.1; IPL PATH = B93E (MN=B93E)
%S.NSI1163  LOCAL DATE = <date>, TIME = <time> FROM SVP (MESZ)
%S.NSI00E3  SLED-REPS READ: 0; EXECUTED: 0
?S.NSD1003  STANDARD SLED ? REPLY (Y; N; EOT=Y)
s.
%S.NSD1000  SLED VERSION <version> LOADED FROM HIP6.1 TO 021D7000
%S.NSI3135  IPL DISK-SETUP READ FROM IPL-CONF PREPARED <date> <time>
%S.NSD1604  WARNING: SLEDSAVE ON VOLUME HIP6.1 TOO SMALL
%S.NSD1111  PRODUCT-ID OF DUMPED SYSTEM: BS2000 <version>
?S.NSD5200  SPECIFY NAME OF SLED PARAMETER FILE. REPLY (NO FILE=EOT; FILENAME; STANDARD
NAME=STD; END)
s.
?S.NSD1113  DO YOU WANT TO CHANGE CURRENT SLED RUNTIME LIMIT OF 045 MINUTES ? REPLY (Y;
N; EOT=N; - (BACKTRACK))
s.
```

```
?S.NSD3000  SPECIFY OUTPUT DEVICE. REPLY (DPUB; DPRIV; TAPE; PRINTER; EOT=DPUB; -
(BACKTRACK))
s.tape
?S.NSD3800  SPECIFY VSN OF SLED OUTPUT TAPE. REPLY (VSN; VSN*; - (BACKTRACK))
s.cs563k
?S.NSD3822  SPECIFY MN OF SLED OUTPUT TAPE CS563K. REPLY (MN;  - (BACKTRACK))
s.me
%S.NSD3810  TAPE CS563K ON DEVICE ME INITIALISED AS T6250
%S.NSD5000  DEFAULT TAKEN: MODE=STD
%S.NSD5000  DEFAULT TAKEN: TASK=STD
%S.NSD1112  SLED RUNTIME LIMIT SET TO   45  MINUTES
%S.NSD1800  STATUS   SECTION : TERMINATED. LAST BLOCK = 8. TIME = <time>
%S.NSD1800  PSA      SECTION : TERMINATED. LAST BLOCK = 18. TIME = <time>
%S.NSD1701  MAINMEM  SECTION : PAGE SELECTION STARTED. TIME = <time>
%S.NSD1702  MAINMEM  SECTION : DUMP STARTED. TIME = <time>
%S.NSD1800  MAINMEM  SECTION : TERMINATED. LAST BLOCK = 80772. TIME = <time>
%S.NSD1702  VM2HYPVS SECTION : DUMP STARTED. TIME = <time>
%S.NSD1800  VM2HYPVS SECTION : TERMINATED. LAST BLOCK = 82442. TIME = <time>
%S.NSD1800  IOHIOSDP SECTION : TERMINATED. LAST BLOCK = 82708. TIME = <time>
%S.NSD1701  PAGEPHYS SECTION : PAGE SELECTION STARTED. TIME = <time>
%S.NSD1702  PAGEPHYS SECTION : DUMP STARTED. TIME = <time>
%S.NSD1800  PAGEPHYS SECTION : TERMINATED. LAST BLOCK = 84776. TIME = <time>
%S.NSD1800  PROTKEYS SECTION : TERMINATED. LAST BLOCK = 84906. TIME = <time>
%S.NSD1800  TSOSCAT  SECTION : TERMINATED. LAST BLOCK = 101074. TIME = <time>
%S.NSD1800  EQUISAMQ SECTION : TERMINATED. LAST BLOCK = 101114. TIME = <time>
%S.NSD1800  SJOBPOOL SECTION : TERMINATED. LAST BLOCK = 101148. TIME = <time>
%S.NSD1800  REPLOG   SECTION : TERMINATED. LAST BLOCK = 102364. TIME = <time>
%S.NSD1800  CONSLOG  SECTION : TERMINATED. LAST BLOCK = 102588. TIME = <time>
%S.NSD1800  HELFILE  SECTION : TERMINATED. LAST BLOCK = 103000. TIME = <time>
%S.NSD1800  SERSLOG  SECTION : TERMINATED. LAST BLOCK = 174636. TIME = <time>
%S.NSD1800  MSCFTRAC SECTION : TERMINATED. LAST BLOCK = 174712. TIME = <time>
%S.NSD1800  SJMSFILE SECTION : TERMINATED. LAST BLOCK = 174782. TIME = <time>
%S.NSD1800  PAGELOG  SECTION : TERMINATED. LAST BLOCK = 174786. TIME = <time>
%S.NSD1800  SLEDMEM  SECTION : TERMINATED. LAST BLOCK = 176308. TIME = <time>
%S.NSD1800  SLEDLOG  SECTION : TERMINATED. LAST BLOCK = 176314. TIME = <time>

%S.NSD1802  SLED OUTPUT COMPLETED
?S.NSD5200  SPECIFY NAME OF SLED PARAMETER FILE. REPLY (NO FILE=EOT; FILENAME; STANDARD
NAME=STD; END)
s.end
%S.NSD1001  SLED TERMINATED
```

10.3 SLED control

SLED can be controlled by various means.

– In attended SLED (= manual SLED), an operator dialog is performed. SLED is controlled either by the statements defined in the parameter file (which is specified by the operator) or by individual parameters entered by the operator.

– In unattended SLED (= automatic SLED), there is no operator dialog. SLED is controlled via the standard SLED parameter file $TSOS.SYSPAR.SLED.<ver> or through the evaluation of default values.

– The use of asynchronous command inputs can output information about the SLED run.

Limiting the SLED runtime

The default value for the runtime limit for SLED is 45 minutes. The default value can be changed in the dialog by replying to the following message with Y:

```
NSD1113   DO YOU WANT TO CHANGE CURRENT SLED RUNTIME LIMIT OF (&OO) MINUTES ?
           REPLY (Y; N; EOT=N; – (BACKTRACK))
```

When Y is entered as the reply, the following message is output:

```
NSD1114   SET RUNTIME LIMIT.
          REPLY  (1–999 (MINUTES); N (CURRENT LIMIT); EOT=N; – (BACKTRACK))
```

A runtime limitation of 1 up to 999 minutes can be specified.

SLED confirms the runtime limit set with message NSD1112.

If the runtime limit is enabled, a check is made for the first time to see if the time limit has expired after main memory has been saved, and after that after every section. If it has expired, then SLED is terminated after the SLEDMEM and SLEDLOG sections have been written.

During the dump of the paging area, the timer is checked after every 2Gbytes have been saved and the save may be terminated prematurely. The dump containing the data in the paging area may also be incomplete.

A premature termination of SLED due to the time limit expiring is announced by the messages NSD1804 and NSD1803.

SLED dialog

Advance setting of parameters

In certain messages of the initial dialog it is possible both to enter a response to the appropriate query and also to set certain parameters in advance, which would otherwise have to be queried in subsequent dialog steps or defined for the default values of SLED.

Example

```
NSD3000 SPECIFY OUTPUT DEVICE.
         REPLY (DPUB; DPRIV; TAPE; PRINTER; EOT=DPUB; – (BACKTRACK))
```

The following responses are possible to the above message:

`TAPE` output to tape.

`TAPE,VSN=vsn`
The VSN of the output medium is made known in advance

`TAPE,VSN=vsn,DEV=mn,MODE=NSF`
The VSN of the output medium and the (partial) scope of the output data are made known in advance

`TAPE,VSN=vsn,DEV=mn,MODE=NSF,TASK=ALL`
All parameters for the output to tape are made known in advance

The syntax is thus the same as for the parameter list of a BS2000 command, where not more than one positional operand and optionally several keyword operands may occur. The permissible combinations of positional and keyword operands can be found in the descriptions of the individual messages.

The input must not contain any blanks. The first blank encountered is treated as the end of the input.

If no positional operand is specified (i.e. either keyword operands only or no operands at all), the default value is assumed.

If a keyword operand is specified without a value (e.g. `MODE=`), the default value is assumed. The appropriate default value is defined by the message via which the operand value would be interrogated if no advance setting is made.

Where parameters are entered in advance, errors may occur which have the effect of the input being ignored. A message is displayed to draw attention to this fact.

Cancellation option

Some of the messages in the initial dialog can be canceled by entering "–" or "––".

Entering "–" causes the last input to be canceled (simple cancellation).

Entering "––" causes all previous inputs to be canceled. The dialog is continued with message NSD5200, the first message in the initial dialog. In the case of a SLED with a parameter file, further processing of the parameter file is terminated after this entry.

In both cases any parameters set in advance are ignored after an appropriate message (NSD5003).

If a message contains a simple cancellation option, this is indicated in the reply section of the message:

```
NSDxxxx ... REPLY (...; – (BACKTRACK))
```

SLED with parameter file

All the instructions needed for running SLED in operator-assisted or automatic mode can be parameterized in the form of a file. This parameter file, which must be created on a public disk, is used to store all the inputs to SLED; the entries are not checked for correct syntax until SLED is running.
The SLED parameter file must not be empty and must possess the following file attributes:

FILE-STRUC=SAM
BUF-LEN=STD or (STD,2)
REC-FORM=V
BLK-CONTR=PAMKEY or DATA

All the parameters for a SLED run must be contained in a record in the parameter file. The individual parameters are separated by commas. The character string must not include any blanks. Each parameter must be prefixed by the appropriate keyword. The sequence of parameters is immaterial. The parameter records must not contain lowercase letters or nonprintable characters.

For an aautomatic SLED run the parameters are read from the parameter file with the name $TSOS.SYSPAR.SLED.<ver>, which means that even in this case flexible control of the dump is assured.

If the software product HSMS is used on the system involved, systems support must ensure that the parameter file is not automatically migrated and thus made inaccessible if it is not used for a long time.

Assignment of the parameter file

The decision as to whether and which parameter file is to be used for the SLED run is made by the operator by entering the following response to message NSD5200:

```
NSD5200   SPECIFY NAME OF SLED PARAMETER FILE.
           REPLY (NO FILE=EOT; FILENAME; STANDARD NAME=STD; END)
```

FILENAME A parameter file is assigned for a SLED run by entering a valid fully qualified file name in response to the message. If the catalog ID or user ID is omitted, the default options (catalog ID of the home pubset and $TSOS) are inserted.

STD By entering STD in response to the message the default SLED parameter file of the system, $TSOS.SYSPAR.SLED.<ver>, is assigned.

EOT (no input)
A null input (EOT) in response to the message means that no parameter file is used. In an operator-attended run, SLED prompts the operator for necessary input to be made from the console.

Processing the parameter file

Parameter records that have already been processed and for which there is already a non-empty output file on magnetic disk are ignored for the next SLED request (in automatic mode without a confirmation prompt, in operator-assisted mode when the following message is answered with N):

```
NSD3204   SLED OUTPUT FILE (&00) IS NOT EMPTY.
           OVERWRITE ? REPLY (YES=Y; NO=N)
```

This makes it possible to react to multiple system interruptions (over a longer period of time) with a parameter file containing several parameter records.
All the parameters for a session must be entered in one line, e.g.

- For output to private disk:
  OUTPUT=DPRIV,FILE=...,MODE=,TASK=STD,VSN=*,DEV=D6
- For output to public disk:
  OUTPUT=DPUB,FILE=...,MODE=,TASK=STD
- For output to an emulated tape:
  OUTPUT=TAPE,VSN=SLED*,DEV=M0,MODE=NSF,TASK=(1EF0,1431,2EE4,5QA1)

End of processing is reached when a SLEDFILE is written or when a parameter record with the character string OUTPUT=END is detected. Subsequent records are ignored and SLED terminates. In automatic SLED mode system loading is initiated.

End-of-file is indicated by an appropriate message at the console.In attended operation SLED issues the NSD5200 message again. In automatic SLED system loading is initiated.

Error behavior in operator-assisted mode (manual SLED)

- If an error occurs during processing of a parameter record as a result of incorrect or missing entries, SLED prompts the operator to enter the required parameter at the console.
  Once the error has been corrected or the missing data supplied, SLED continues processing the file.
  If the operator decides on the cancel option, further processing is aborted.
- If, during processing, empty records or records containing lowercase characters or non-printable characters are detected, SLED issues message `NSD5245` asking the operator how it should proceed.
  Processing of this parameter file can either be aborted or continued with the next record.

Error behavior in unattended mode (automatic SLED)

- If the $TSOS.SYSPAR.SLED.<ver> file does not exist on the home pubset or SLED cannot find it (e.g. because TSOSCAT has been destroyed), SLED proceeds as for automatic SLED without a parameter file (see page 389.)
- If the parameter file does not have the required file attributes, it is rejected and the system switches over to operator-assisted mode.
- If an error occurs during processing of a parameter record as a result of incorrect entries or a lack of data, processing of the parameter file ceases and the system switches over to operator-assisted mode.
- Empty parameter records of a parameter file are ignored.
- If no values are specified for the MODE or TASK parameter, SLED determines these values itself.

Automatic SLED

Automatic SLED enables a memory dump to be taken without operator intervention and the system to be subsequently reloaded.

SLED is loaded in automatic mode if the “Automatic Restart” function is switched on after a system crash and SLED is set as the dump generator (see the command SET-RESTART-OPTIONS MODE=*ON(...), DUMP=*SLED). Automatic SLED is loaded from the SYSRES belonging to the home pubset of the aborted system run. Information on the automatic restart control can be output with the SHOW-RESTART-OPTIONS command.

There are two modes in which an automatic SLED can take place:

1. Parameter file

   First, SLED looks for the default parameter file $TSOS.SYSPAR.SLED.<ver> on the home pubset. When this file has been found and if it can be processed, SLED uses the entries specified there.
   Parameter records that write to non-empty disk files are ignored.

2. Use of default values

   If SLED does not find the standard parameter file $TSOS.SYSPAR.SLED.<ver>, the following default definitions are taken for executing the automatic SLED:

   – dump file: `$TSOS.SLEDFILE`
   – MODE parameter: `EOT`
   – TASK parameter: `EOT`

   This corresponds to the following statements in the parameter file:

```
OUTPUT=DPUB,FILE=$TSOS.SLEDFILE,MODE=,TASK=
OUTPUT=END
```

Error condition

In the event of a serious error, SLED switches from automatic SLED to attended operation. This also applies if a parameter file is being used and missing or errored entries are detected during processing of a parameter record.
In the event of a disk availability error during an automatic SLED run (message `NSD1400`), the data involved is not available to SLED.

Conditions for execution without operator intervention

- The SLED version must match that of the operating system.
- All disk devices on which public disks or paging disks are mounted must be ready for operation.
- The file $TSOS.SLEDFILE or the file assigned in the parameter file
  - must be set up
  - must have been created on the home pubset (for $TSOS.SLEDFILE only)
  - must not be protected against write access
  - must not be protected by a password
  - must be at least twice the size of main memory
  - must be logically empty
  - must have reached the expiration date
  - must not be on DRV disks.

Automatic system restart

Automatic system restart after an automatic SLED is assured in the following cases:

- The automatic SLED has been completed without errors and all the accessible data could be written to a SLED output file.
- The automatic SLED works without a parameter file and output file $TSOS.SLEDFILE was not empty. In this case, no data is dumped and the system is reloaded immediately.
- For an automatic SLED with a parameter file, non-empty output files on magnetic disk are ignored until a SLEDFILE has been created or the end of the parameter file has been reached or a parameter record containing the character string `OUTPUT=END` is detected.

  This enables more than one SLED output file to be made available which can then be written to one after the other in different SLED runs.
- The parameter file contains only one parameter record with the instruction `OUTPUT=END` The operator has the option in this case of initiating an automatic system restart without creating diagnostic data.

Asynchronous command inputs

In the case of SLED, an asynchronous input is any input made from the console which is not a response to a message. Analogous to the BS2000 commands all asynchronous commands begin with a slash (/).

These commands cannot be entered at any desired point within the SLED run. Generally they are not permitted unless the start dialog has been completed and processing of the dump has begun.

The following asynchronous input is processed; all others are rejected with an appropriate message.

STATUS command

This command provides the operator with information on the status of the SLED run.

The last block that has been written by SLED is output. This gives the operator the opportunity to determine how far the SLED run has progressed, even while a section is still being processed.

SLED messages

SLED messages begin with the message code `NSDxxxx`.

The number designated by "`xxxx`" has the following possible meanings:

09xx	Execution of non-standard SLED	
1xxx	Execution:	
	10xx	Start and termination
	11xx	Inconsistencies
	12xx	Internal errors
	14xx	Disk availability
	16xx	Disk access
	18xx	Scope of dump
	19xx	Automatic SLED
3xxx	Output:	
	30xx	General
	32xx	Disk file
	33xx	Disk error
	34xx	Private disk
	36xx	Public disks
	38xx	Tape
	39xx	Tape error
5xxx	Input processing:	
	50xx	Advance setting of parameters
	52xx	Parameter file
	55xx	Specific checks
	56xx	Asynchronous inputs
7xxx	Sections of SLED output:	
	72xx	Main memory
	73xx	HSA
	74xx	PSA
	76xx	System files
	78xx	Virtual areas
	79xx	SLEDLOG

10.4 Extracting IOHDUMP from a SLED

On x86 servers, the IOHDUMP diagnostic data is written to the BS2000 SLED file, where it is stored in the IOHIOSDP section. This data is used to diagnose HSI errors. It can be processed as follows:

```
/START-DAMP
//OPEN-DIAGNOSIS-OBJECT OBJECT <sledfile>
//SHOW-EDITED-INFORMATION INFORMATION=*DUMPED-SYSTEM-FILE ————————————  (1)
//END
/SHOW-FILE-ATTR *IOHIOSDP*,CREATION-DATE=*TODAY ——————————————————————  (2)
```

(1) The IOHIOSDP section is displayed. The only function that may be used on it is “GEN”. Marking this function and pressing the DUE key causes DAMP to extract this data and write it to a PAM file in BS2000. The name of this file is shown in the first two lines of the DAMP screen.

(2) The name of the file generated is displayed.

You can then transfer this file from BS2000 to X2000 as follows:

```
/START-FTP
open <system> ————————————————————————————————————————————————————  (1)
<userid> —————————————————————————————————————————————————————————  (2)
<password>———————————————————————————————————————————————————————  (3)
bin———————————————————————————————————————————————————————————————  (4)
put <filename> <tar_archive_name>————————————————————————————————  (5)
quit
```

(1) Create a connection to the X2000 system, specifying the systems IP address or symbolic name.

(2) Specify the user ID of the X2000 system to which the data is to be transferred.

(3) Enter the password for this user ID.

(4) The PAM file has to be transferred in **binary** mode to avoid corrupting the file structure.

(5) Transfer the file `<filename>` to the file `<tar_archive_name>` on the target system.

`<tar_archive_name>` must not contain a catalog ID or user ID.

You need to complete the following steps in order to process the file under X2000:

```
tar tf <tar_archive_name> ————————————————————————————————— (1)
tar xf <tar_archive_name> <file_name> ————————————————————— (2)
uncompress <file_name> ———————————————————————————————————— (3)
```

(1) List the contents of the tar archive with the name `<tar_archive_name>`.

(2) Choose the file you want and extract it from the tar archive.

(3) If necessary, uncompress the diagnostic data.

You can then process the diagnostic data in the usual way:

- You can use MDEBUG to work with IOHDUMP (you open IOHDUMP as a dump file with `sd`):
- SYSDB Trace and PRKDUMP can be opened with `exs` and with `exp`, respectively; you can use special MDEBUG statements to access the IOH data.

11 SNAP dump

The SNAP dump generator bridges the gap between the dump generators SLED and CDUMP. SNAP is part of BS2000 OSD/BC.

During dump generation by CDUMP, the operating system continues to run and may thus “corrupt” the data to be included in the dump. SLED terminates the operating system, so you need a relatively long time to dump the data and restart the system.

SNAP interrupts the operating system for not more than 24 seconds, dumps specific memory areas (see below) and then restarts the operating system. SNAP operates independently of BS2000 and therefore does not corrupt the diagnostic data involved in the dump.

SNAP is called by the operating system from the TPR or SIH status via the $SNAP interface. It is generally called whenever there is an inconsistency in the operating system status that is, however, not serious enough to cause the session to be aborted.

The SNAPTIME system parameter can be used to control when SNAP returns control to BS2000. The default value is 24 seconds. This is the maximum value with SNAP, since the system state “BS2000 terminated” could otherwise occur. Depending on SNAPTIME, the size of the SNAP dump is limited. The maximum dump size depends primarily on the data transfer rate of the I/O and on the speed of the disk containing the file $TSOS.SNAPFILE. With the servers and disks supported by SNAP, the maximum size of a SNAP dump is 1 GB.

A SNAP dump contains the following data:

- class 1, class 3 and, optionally, resident[1] class 4 memory

  In BS2000/OSD-BC V9.0 and higher main memory above the 2 GB boundary and thus, possibly, the entire resident class 4 memory is also contained in the SNAP dump.

- name list and entry point list of all operating system modules (EOLDTAB)
- administration data
- hardware status register of the current virtual machine

[1] Here, resident means that the page is located in main memory.

The SNAP run is logged in the form of messages at the console.
The messages of SNAP have the message class NSP. Information on individual messages can be obtained in ongoing operation with the HELP-MSG-INFORMATION command.

In particular, the SNAP function is also used by the CDUMP dump generator to create a consistent (uncorrupted) backup of class 1, class 3 and resident class 4 memory.

11.1 SNAP files

In order for SNAP to run, a system file with the name SNAPFILE must be created on the home pubset. It must be at least 16 MB in size and not larger than 1 GB. A value of at least 144 MB is recommended. It must not be copied from another pubset.

The SNAPFILE is installed during system generation using the SIR untility routine or when SNAP is activated using the ACTIVATE-SNAPSHOT command. It is cataloged under the TSOS user ID. During ongoing operation changes may be made to the SNAPFILE (configuration, changing size, deletion) only using the ACTIVATE-SNAPSHOT and DEACTIVATE-SNAPSHOT commands, see the “Commands” manual [8].

If the SNAPFILE is not available, it is created in the standard size when startup takes place (SNAP-ACTIVE-SWITCH=ON parameter) or in the specified size when the ACTIVATE-SNAPSHOT command is executed.

SNAP writes the diagnostic data to the SNAPFILE. To make sure that this file is emptied and thus available for subsequent SNAP calls, the SNAPFILE contents are automatically written to a dynamically generated system file after BS2000 is reactivated. This operation is given a high priority. These dump files are cataloged under the user ID SYSSNAP and are given the name

```
SNAP.snap-id.date.time
```

where:

SNAP	identification as SNAP output
snap-id	7-character ID from the associated SNAP call
date	date in the form: yyyy.mm.dd
time	time in the form: hh.mm.ss

The user ID SYSSNAP is created automatically during first startup.
The disk storage available for this ID should be at least twice as large as $TSOS.SNAPFILE.

The file $TSOS.SNAPFILE cannot be processed directly by any dump analysis routine. However, transferring the data from $TSOS.SNAPFILE to $SYSSNAP creates a file format that can be analyzed by the DAMP dump analysis routine. If the system is no longer able to transfer the file $TSOS.SNAPFILE to $SYSSNAP, $TSOS.SNAPFILE is automatically saved to an editable file under $SYSSNAP the next time the system is started up or at ACTIVATE-SNAPSHOT. In the case of a DRV home pubset, it is possible that the SNAPFILE can no longer be converted in the next session, and this half-finished SNAP dump is then lost.

The SNAPFILE file is discarded if it was created in the previous session within the framework of a system dump (CDUMP).

The SNAPFILE is only copied or converted when the SNAP function is activated.

No further SNAP calls are accepted until the SNAPFILE is emptied.

11.2 Activating and deactivating SNAP

The SNAP-ACTIVE-SWITCH=ON/OFF parameter in the startup parameter service specifies whether SNAP is activated immediately in the current session, see the “Introduction to System Administration” [6]. When SNAP-ACTIVE-SWITCH=OFF, SNAP is initially not available for this session. SNAP calls are terminated with a corresponding return code. SNAP can later be activated or deactivated dynamically again as often as required using the ACTIVATE- and DEACTIVATE-SNAPSHOT commands.

The commands for SNAP are described in detail in the “Commands” manual [8].

Command	Meaning
ACTIVATE-SNAPSHOT	Activates dump generator SNAP
DEACTIVATE-SNAPSHOT	Deactivates dump generator SNAP
SHOW-SNAPSHOT-STATUS	Outputs information on SNAP

Table 21: Command overview for SNAP

The ACTIVATE- and DEACTIVATE-SNAPSHOT commands are executed asynchronously. Message NSP4000 confirms correct acceptance of the command. The SHOW-SNAPSHOT-STATUS command enables you to check the changed settings.

11.3 Restrictions

SNAP runs on all BS2000 servers.

The data saved thus far is preserved and the fragmented SNAP dump is made available to the diagnostic staff in the following cases

- the data to be saved by SNAP (maximum 1 GB) cannot be written within the time set for SNAPTIME (Default: 24 seconds)
- the SNAPFILE file is too small
- an internal error occurs

In this case, message `NSP1010` is output. For serious errors in SNAP, the SNAP process is aborted.

Within BS2000 there are certain facilities which recognize if the BS2000 system has been inoperable for longer than a predefined time. SNAP deactivates BS2000 so that it can secure diagnostic data under its own runtime control. However, this does not been that BS2000 is “dead”. When defining intervals for the detection of system failures, it should be borne in mind that SNAP-EXEC is still active (as a “stand-in”) even though BS2000 is inactive.

In this context, special attention is drawn to the product HIPLEX MSCF, which can be used for vital-sign monitoring for BS2000 systems in conjunction with the MSCF configuration parameter FAIL-DETECTION-LIMIT.

BS2000 OSD/BC and SNAP support a working memory of more than 2 GB.
As a result, not only the class 1 and class 3 memory may be contained in the SNAP dump, but also the entire resident class 4 memory if the time limit set in the system parameter SNAPTIME is not exceeded and the SNAPFILE file is sufficiently large.
The system parameter SNAPTIME is limited to the interval of 8 to 24 seconds. When too low a value is set, it is rounded up to 8 seconds, and when too high a value is set, it is rounded down to 24.

11.4 Automatic SNAP

The SNAP is activated in automatic mode if the “Automatic Restart” function is active at the time of a system crash and SNAP is set as the dump generator (see the command SET-RESTART-OPTIONS MODE=*ON(..., DUMP=*SNAP)).

Automatic SNAP makes it possible to create a memory dump without operator intervention and then reload the system. Compared to automatic SLED, automatic SNAP has the advantage that the system can be restarted after a short time but that, despite this, the most important diagnostic data is saved.

The SHOW-RESTART-OPTIONS command can be used to output information relating to the control of the automatic restart.

12 TRACE MANAGER
Collect diagnostic information during the session

The TRACE MANAGER collects all decentrally created component traces, a “trace” being defined as the collection of diagnostic information, during a session, at previously defined discrete locations in the system. This information consists of consecutive, individual records containing data specific to a component or the system, such as system status variables, program data, parameter lists, time stamps, etc. Traces are also saved when a dump is taken, thus improving the facilities for BS2000 system diagnosis as part of dump evaluation.

Traces may be performed locally (for a task) or globally (for the entire system). Consequently, the memory areas for storing the information (trace buffers) are located in either the task address space or the system address space. A local trace has a trace buffer and associated management data within each task. A global trace requires only one buffer and one set of management data for the entire system.
The TRACE MANAGER does not itself manage the trace buffers; it simply holds pointers to the buffers in its management tables. For this reason, the trace owners must inform the TRACE MANAGER of the size and location of the buffers. Since traces tend to generate large amounts of data in a very short time, the trace buffers are overwritten cyclically. The data in the buffers is saved using SLED or CDUMP and analyzed using DAMP.

In the course of a dump using CDUMP, the TRACE MANAGER generates the trace dump list (TDL), which contains buffer descriptors and selected trace description data (trace ID and the address of the next or last entry). The TRACE MANAGER cannot generate this list for a dump produced with the aid of SLED, since the operating system is no longer running after SLED has been loaded. In this case, the DAMP analysis program handles this task.

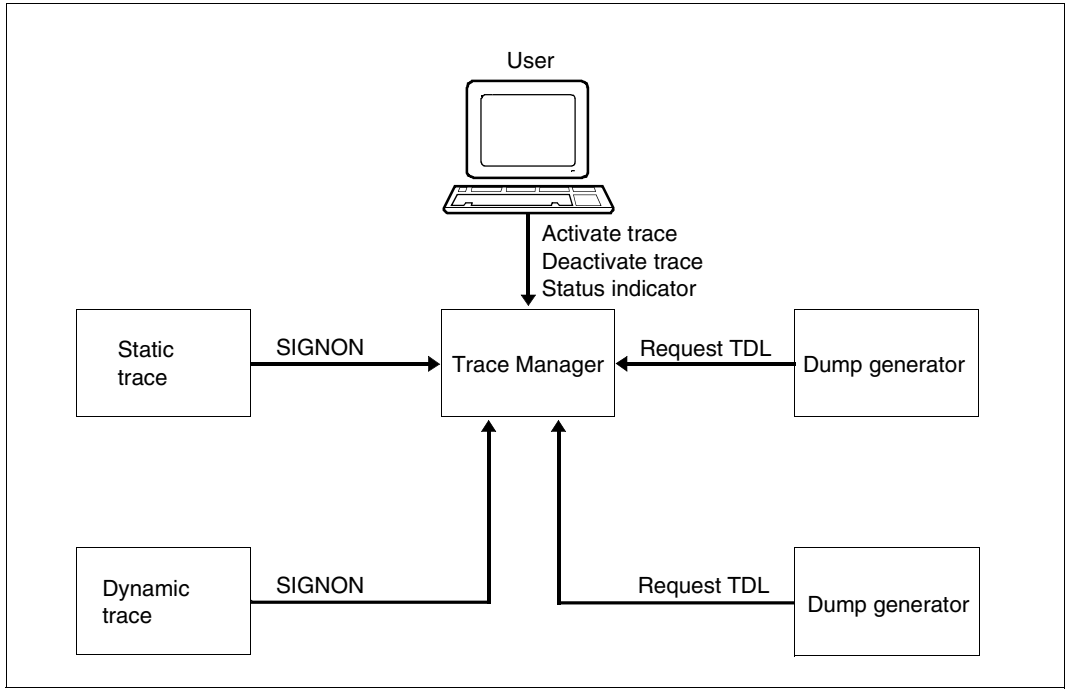


Figure 101: TRACE MANAGER functions

In the context of the central management of traces, various data is kept in tables under the control of the TRACE MANAGER. This data can be divided into three groups with different functions, which are kept in different tables. The management data comprises:

– address lists,
– operating data and
– descriptive data.

The figure 102 on page 403 shows how these data and tables are linked together.

The address lists form the basis of the management tables. The address list for global traces is anchored in the central BS2000 table XVT, while an address list for a local trace is anchored in the TCB of the relevant task. For each trace, the address list contains one fixed-length (8-byte) record which contains the address of the operational data block and a number of trace status flags. The address list is preceded by a list header containing the data describing the table, together with the address of the TDL.

Each defined trace receives an entry in the TDL, which includes, among other things, the address of the trace buffer, the name of the trace and the subsystem, and also the version of the subsystem.

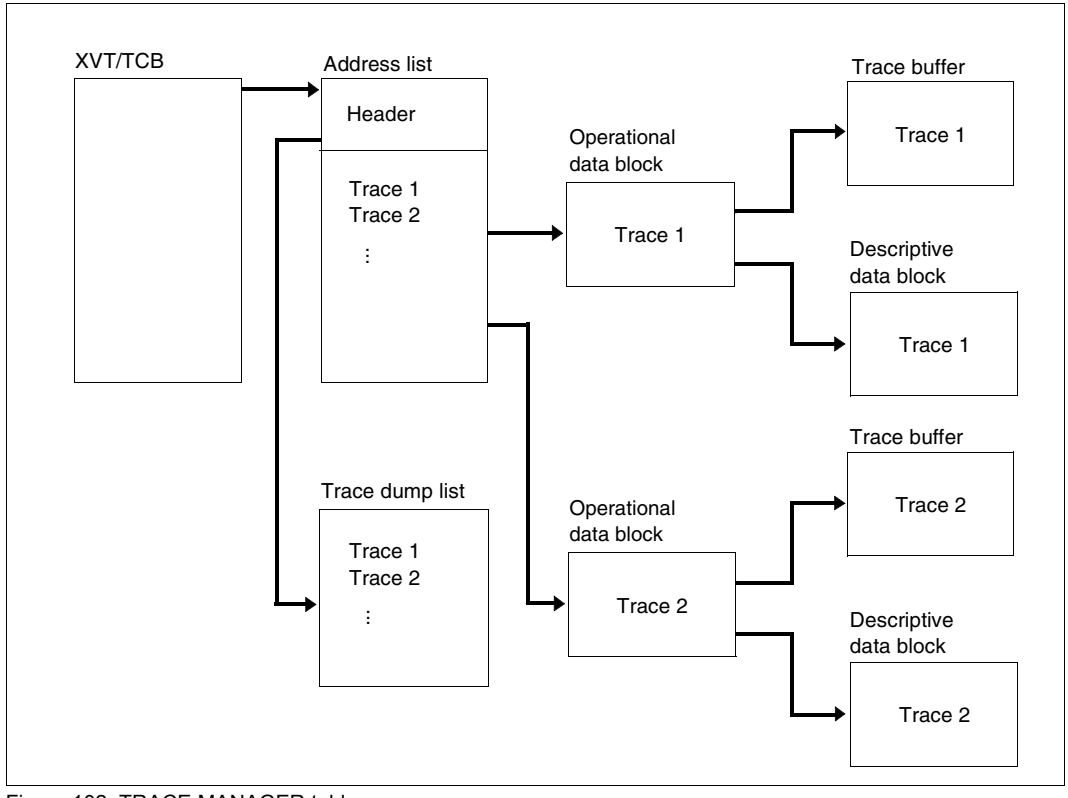


Figure 102: TRACE MANAGER tables

The operational data block contains all the data which is required to access the trace buffer or the next free record in this buffer in the most efficient manner (buffer descriptors). The operational data block contains the address of the corresponding descriptive data block which is chained to the operational data block. This contains all the data needed for management by the TRACE MANAGER and for editing the trace buffer, such as the description of the trace buffer layout, the trace ID etc.

The global trace management tables are set up during the system startup phase: the TRACE MANAGER is called and initializes all the tables needed in connection with the management of global traces.
It is possible to define new traces dynamically and initialize them using the appropriate macros after the system has been started up. To do this, the trace management tables are passed a fixed-length dummy entry for every initialized trace, thus allowing new traces to be accepted (≙dynamically defined).

When setting up local trace management tables, the TRACE MANAGER is called by the base system each time a new task is created. Upon termination of the task, the TRACE MANAGER is called again, this time to release the local trace management tables.

During a system restart, the local task management tables are deleted and replaced by the tables stored at the time of the checkpoint.
Local task traces can also be dynamically defined and initialized after the system has been started up. The global system trace address list has spare entries for the local task traces. The memory required for the local task trace tables is initialized when the tables are installed.

It is also possible to suppress traces dynamically in the subsystem's shutdown routine as well as to define traces dynamically.

In order to define a trace, the TRACE MANAGER requires information, which it stores in the global system management tables. This information can be passed to the TRACE MANAGER either statically (data is already available as code in the trace tables) or dynamically (using a macro).

The TRACE MANAGER is controlled via the following operator or system administrator commands. The commands are described in detail in the "Commands" manual [8].

Command	**Description**
SHOW-TRACE-STATUS	Display information on system traces
START-TRACE	Activate a trace
STOP-TRACE	Deactivate a trace

Table 22: Overview of the TRACE-MANAGER commands

13 Online maintenance

Online maintenance comprises the following functions:

– Storage and analysis of the hardware error statistics file
(see the ELSA utility routine in the “ELSA” [3] manual).

– execution of statistics and trace routines under BS2000 control in parallel with user programs.

Online maintenance in BS2000 is carried out under a user ID which has the HARDWARE-MAINTENANCE privilege. By default, this is the SERVICE user ID, which is set up with the account number of the same name during first start.

In addition, every file owner has the option of controlling access to shareable files for the user ID with the HARDWARE-MAINTENANCE privilege on a file-by-file basis.
This can be done using the `USER-ACCESS` operand of the `/MODIFY-FILE-ATTRIBUTES` command (or CATAL macro). If the access authorization SPECIAL is assigned to a file, that file can be accessed by all user IDs, **including** the user ID with the HARDWARE-MAINTENANCE privilege.

If programs catalogued under the SERVICE user ID (e.g. ELSA) are to be executed under a user ID with the HARDWARE-MAINTENANCE system privilege, likewise a `/MODIFY-FILE-ATTRIBUTES` command with the operand `USER-ACCESS=*SPECIAL` must be issued for these routines.
This access authorization is canceled by means of another `/MODIFY-FILE-ATTRIBUTES` command (or CATAL macro) using the operand `USER-ACCESS=*ALL / *OWNER-ONLY`.

14 Error files and logging files

Error files and logging files are useful tools for systems support for detecting hardware faults or software errors and for monitoring the entire message traffic between consoles, authorized user programs and the system.

The error files SERSLOG and HEL and the logging files CONSLOG and RESLOG are described.

The data supplied by the error files and logging files is evaluated by the diagnostic functions and programs.

14.1 Hardware error logging file HEL

The Error Logging System ELS records the following events for diagnostic purposes:

- Error events
  - machine errors
  - errors related to I/O interrupts
  - errors in peripheral equipment
- other events
  - statistical information from peripheral equipment
  - statistical information from the CPU firmware
  - processor context saves
  - information from test and diagnostic programs (TDP)

The events are recorded by the HEL system task and documented in the central statistics file which is automatically created. The name of the HEL file has the following structure (yyyy-mm-dd.hhmmss: creation date and time):

$TSOS.SYS.HEL.yyyy-mm-dd.hhmmss

Evaluation of the HEL files is performed by the BS2000 utility routine ELSA.

The HEL task and the BS2000 evaluation program (ELSA) can access the statistics file at the same time. ELSA is described in detail in the “ELSA” manual [3].

Service who is logged on under a user ID which has the HARDWARE-MAINTENANCE privilege can use the following commands to control the saving of HEL records as well as to control and support monitoring by Remote Service:

Command	Meaning
CHANGE-HEL-FILE	Close the current HEL file and opens a new file
MODIFY-HEL-CHECK	Control threshold monitoring
MODIFY-HEL-LOGGING	Control saving of HEL records
MODIFY-HEL-TELESERVICE- ALARM	Set the error threshold values for Remote Service messages
SHOW-HEL-CHECK	Request information about threshold value monitoring
SHOW-HEL-LOGGING	Request information about the logging records
SHOW-HEL-STATUS	Request general information about hardware error logging
SHOW-HEL-TELESERVICE-ALARM	Request information about the Remote Service parameters which are set
START-HEL-LOGGING	Activate the hardware error logging system function and opens the logging file
STOP-HEL-LOGGING	Terminate the hardware error logging system function and closes the logging file

Under the TSOS user ID, systems support can use the `/SHOW-HEL-STATUS`, `/START-` and `/STOP-HEL-LOGGING` commands.

The commands are described in detail in the “Commands” manual [8].

14.2 Software error logging file SERSLOG

The software error logging function SERSLOG has been provided to facilitate the diagnosis of BS2000 errors. BS2000 error information can be written to the SERSLOG file by various routines.
During startup, the SERSLOG file is opened and error logging activated. The name of the SERSLOG file has the following format:

SYS.SERSLOG.yyyy-mm-dd.xxx.nn

yyyy-mm-dd	Date on which the file was opened
xxx	Number of the current session
nn	Serial number of the SERSLOG file (01-99; always 01 at startup).

The SERSLOG file is not write-protected.

During shutdown, the SERSLOG file is closed and error logging terminated.
The current (or most recent) SERSLOG file is included in the SLED dump.

Error logging can only be controlled by the operator and systems support, who can switch the SERSLOG file and activate or deactivate error logging.
When nn is greater than 99, the counter is reset to 01, thus overwriting the first SERSLOG file if the date and the session number are the same.

Command	Meaning
CHANGE-SERSLOG-FILE	Close the current SERSLOG file and a open new file
SHOW-SERSLOG-STATUS	Request information on the state of SERSLOG and the name of the logging file
START-SERSLOG	Activate the SERSLOG function
STOP-SERSLOG	Terminate the SERSLOG function

Automatic monitoring of critical system statuses which are reflected in SERSLOG events is possible with ASE. For details, see the chapter “ASE Auxiliary SERSLOG Extensions” on page 365.

14.3 CONSLOG logging file

All the messages exchanged between consoles, authorized user programs and the system are recorded in a logging file. This excludes the last messages of a session which are output when the home pubset is written back to disk. In addition to the console dialog, the CHANGE-CONSLOG-FILE command is also logged when entered from a data display terminal.

Command	Meaning
CHANGE-CONSLOG-FILE	Close the current logging file and open a new one
SHOW-CONSLOG-ATTRIBUTES	Determine the status of system logging and the name of the logging file
SET-CONSLOG-READ-MARK	Enable read access to the current CONSLOG file without having to close it first.
TURN	Evaluate the current logging file

Logging is activated automatically during system initialization.
A message indicates which logging file has been opened.

The file name of the logging file depends on the system parameters NBKESNR and FMTYFNLG. These parameters allow the following file naming conventions to be specified for the SYSAUDIT or TSOS IDs:

New format: SYS.CONSLOG.yy.mm.dd.xxx.nnn
SYS.CONSLOG.yyyy-mm-dd.xxx.nnn

Old format: SYS.CONSLOG.yy.mm.dd.xxx.nn
SYS.CONSLOG.yyyy-mm-dd.xxx.nn

where:

yy.mm.dd / yyyy-mm-dd	New format: date on which the file was opened Old format: date on which the session started
xxx	Number of the current session
nnn	Serial number of the logging file (001-999 for each day on which the CONSLOG file was switched)
nn	Serial number of the logging file (01-99 per session)

With the aid of the system parameter NBKESNR systems support can define whether the CONSLOG file is cataloged under the user ID TSOS or under SYSAUDIT, and whether a total of 99 CONSLOG files per session or 999 per day can be created.

By using the system parameter FMTYFNLG systems support can define whether the year in the name of the logging file is stored as two digits (omitting the century, in the format yy.mm.dd) or alternatively is in four-digit form (including the century, in the format yyyy-mm-dd).

If an unrecoverable DMS error occurs during logging, then the current logging file is closed and a new one opened with the serial number +1.

If error message

```
DMS0541   INSUFFICIENT SPACE ON DISK ....
```

is output and

– the system parameter NBLOGENF (force console logging) is set, then the operator also receives the message

  ```
  NBR0953  ERROR DURING CONSOLE-LOGGING PROCESSING. REPLY (R=RETRY; H=HALT).
  ```

  If memory space is created under the ID under which the CONSLOG file is stored (TSOS or SYSAUDIT) then the query may be answered with "R". Console logging is continued with the logging file serial number +1. Data records are not lost. If the operator enters an "H", then the session is ended.

– the system parameter NBLOGENF (force console logging) is not set, then console logging is deactivated. Message NBR0906 notifies the operator that console logging is no longer active. If memory space is now created under the relevant ID, console logging may be reactivated with the CHANGE-CONSLOG-FILE command. Data records up to the time that console logging is reactivated are lost. In order to indicate that logging is incomplete, the serial number of the new logging file is incremented by 2.

Messages output from the SYS.CONSLOG file by TURN processing are not included in the logging file in order to avoid multiple logging.

Systems support also has the option of closing the current logging file during the session and opening a new logging file.
System administration can use the SET-CONSLOG-READ-MARK command to make the current open logging file readable so that it can be copied, for example.

All closed logging files can be accessed during the session (e.g. PRINT-DOCUMENT command).

The maximum possible number of logging files during a session is 99 if the sequential number of the logging file has been specified as two digits, or 999 per day if a three-digit sequential number has been specified. There will be no automatic change of the logging file (e.g. when the date changes). If the maximum number is exceeded, a message will be set to the operator and nothing more is written to the logging file until the end of the session. It is possible to prevent the maximum number of logging files being exceeded by means of the system parameter NBLOGENF.

A short date record is written at intervals of approx. 25 records. If the date changes during a session, this is shown in the logging file by the record type “change of day”.
When changing from winter time to summer time and vice versa, a change-of-day record with the new season information is written to the logging file.
This record type is also entered as the first and last record in the logging file. It then contains the date on which the file was opened or closed, and the number of CONSLOG files opened in this session. If the logging file is interrupted due to an error, there is no assurance that the change-of-day record will appear as the last record.
In the last logging file of a session, no final change-of-day record is written. The exportation of the home pubset cannot be logged in the CONSLOG file any more.

The name of the current logging file can be output with the SHOW-CONSLOG-ATTRIBUTES command.

Structure of the logging file

CONSLOG files are always created as SAM files.

Format of a record:

Message

Recipient	Blank	Message type	Sender	-	Job ID	. or # 1)	Time of day	Blank 6)	Text
1-4	5	6	7-10	11	12-14	15	16-21	22	23...

Change-of-day record

Blank	T 2)	CLOG 3)	Blank	. or # 1)	Time of day	Blank	***	Date 4)	***	Blank
1-5	6	7-10	11-14	15	16-21	22-23	24-26	27-36	37-39	40

Continued:

Number of CONSLOG per session	Blank	***	Blank	Time zone 5)	Blank	***...***
41-46	47	48-51	52	53-61	62	63-128

1) The system parameter SECSTART determines whether the separator is a period (default setting) or a # (see the “Introduction to System Administration” manual [6]).
2) T is the identifier for the change-of-day record.
3) CLOG is the task that writes the messages of all the consoles to the logging file
4) date format: yyyy-mm-dd or **yy.mm.dd
(dependent on the system parameter FMTYFLNG, see the “Introduction to System Administration” manual [6])
5) Difference between local time and UTC in hours and minutes; format of time zone specification: UTC±hh:mm
6) Position 22 of the “response” message type contains a period or colon (. or :).

Date record

Year	-	Month	-	Day
1-4	5	6-7	8	9-10

At intervals of 25 records, a date record is entered in the file.
The above representation applies only if the system parameter FMTYFNLG has the value 4; if FMTYFNLG = 2 is specified, the year is only two digits long, and the date record is correspondingly 2 bytes shorter.

The following entries are allowed for “recipient” and “sender”:

– mnemonic name of console, in parentheses
– name of application
– routing code (recipient only)
– task sequence number (TSN) of a user or system task, e.g. of an OPRT

The following entries are possible for “message type”:

% system message which does not require a response
? system message requiring a response, which can also be issued by the operator
& request for additional information, requiring a response from the user who issued the command
; system message requiring a response, which only a task can give
+ result of a command

! command termination message

* error message

E emergency message (message, query or response to an emergency query)

R response to a query (message type ? or &)

/ command

Logging file analysis and backup

The duties of systems support include the analysis and saving of logging files from preceding sessions.
As the logging file is cataloged as a SAM file, it can be analyzed, for example, by means of EDT procedures.

The current logging file can also be evaluated using the TURN command.
Messages can be selected on the basis of different criteria:

– day
– time
– destination
– source.

If the SECOS software product is used, the SATUT component can be applied to analyze CONSLOG files as well. To this end the CONSLOG messages are converted into a SAT logging data record. The code for the event type is always CLG
(see the “SECOS” manual [9]).

Extracts from the logging file

Extract 1: Fetching system information

```
OPRT /(CB)-OOO.133328 SH-SYS-INF
(CB) +XACK-OOO.133328  CONFIGURATION                    = 390SU- 7OO-2O
(CB) +XACK-OOO.133328  CPU-ID-LIST :    ADR   0        = 1102000621600000
(CB) +XACK-OOO.133328                   ADR   1        = 1112000621600000
(CB) +XACK-OOO.133328                   ADR   2        = 1122000621600000
(CB) +XACK-OOO.133328  HSI-ATT :        TYPE           = IX
(CB) +XACK-OOO.133328                   ASF            = YES
(CB) +XACK-OOO.133328                   OPERATION-MODE = VM2OOO
(CB) +XACK-OOO.133328  MEMORY-SIZE                     = 512 MB
(CB) +XACK-OOO.133328  MINIMAL-MEMORY-SIZE             = 256 MB
(CB) +XACK-OOO.133328  BS2OOO-ID :      NAME           = I11BXS
(CB) +XACK-OOO.133328                   VERSION        = V20.0A0OI1
(CB) +XACK-OOO.133328                   OSD-BC-VERSION = V11.0A0OOO
(CB) +XACK-OOO.133328                   UGEN-TIME      = <date> <time>
(CB) +XACK-OOO.133328  IOCONF-ID :      NAME           = SU39OOO1
(CB) +XACK-OOO.133328                   VERSION        = V20.0A0O
(CB) +XACK-OOO.133328                   UGEN-TIME      = <date> <time>
(CB) +XACK-OOO.133328                   FORMAT         = IORSFO1
(CB) +XACK-OOO.133328  IPL-TIME                        = <date> <time>
(CB) +XACK-OOO.133328  SYSTEM-CONF :    SYSID          = 129
(CB) +XACK-OOO.133328                   HOME-PUBSET    = SBZ7
(CB) +XACK-OOO.133328                   HOST-NAME      = DO17ZE15
(CB) +XACK-OOO.133328                   VM-INDEX       = 11
(CB) +XACK-OOO.133328                   VM-NAME        = VM11SU39
(CB) +XACK-OOO.133328                   SYSTEM-NAME    = *NONE
(CB) +XACK-OOO.133328                   SYSPAR-BS2-SEL = *STD
(CB) +XACK-OOO.133328                   LIVE-MIG-COUNT = O
(CB) +XACK-OOO.133328  VM2OOO-VERSION                  = V11.5A
(CB) +XACK-OOO.133328  VM2OOO-MONITOR-  OSD-BC-VERSION = V11.5A0OOO
(CB) +XACK-OOO.133328     SYSTEM:       HOST-NAME      = DO17ZE14
(CB) +XACK-OOO.133328  SYSTEM-TIME-     ZONE           = +01:OO
(CB) +XACK-OOO.133328     PARAMETER:    SEASON         = S
(CB) +XACK-OOO.133328                   SEASON-DIFFERENCE = O1:OO
(CB) +XACK-OOO.133328                   PREV-CHANGE-DATE  = <date> <time>
(CB) +XACK-OOO.133328                   NEXT-CHANGE-DATE  = <date> <time>

(CB) +XACK-OOO.133328                   SYNCHRONIZATION   = SERVER-CONN-EXT-REF
(CB) +XACK-OOO.133328                   EPOCH             = O4
(CB) ! UCO-OOO.133328  %  NBRO74O COMMAND COMPLETED 'SH-SYS-INF';
                          (RESULT: SC2=OOO, SC1=OOO, MC=CMDOOO1); DATE: <date>
```

Extract 2: Issuing various operator commands

```
:
OPRT /(CB)-000.110956 SHMSG
  <* % UCO-000.110956  %  NBR0970 OPERATOR TASK WITH TSN 'XACK' CREATED FOR
                          CONSOLE '(CB)'
(CB) +XACK-000.110956  %  NBR0031 NO MESSAGE OUTSTANDING ON THE CONSOLE
(CB) ! UCO-000.110956  %  NBR0740 COMMAND COMPLETED 'SHMSG';
                          (RESULT: SC2=001, SC1=000, MC=CMD0001); DATE: <date>
  <E %XACL-000.111003  %  NBR0797 APPLICATION '@001' CONNECTED WITH '$CONSOLE',
                          PROCESSOR NAME 'D017ZE14', STATION NAME 'OMS00062'
2008-10-18
OPRT /@001-000.111005 REQ-OPER-ROLE SYSADM
  <* % UCO-000.111005  %  NBR0970 OPERATOR TASK WITH TSN 'XACM' CREATED FOR
                          CONSOLE '@001'
@001 +XACM-000.111005  %  NBR0980 OPERATOR ROLE 'SYSADM' ASSIGNED TO
                          OPERATOR ID 'SYSOPR'
@001 ! UCO-000.111005  %  NBR0740 COMMAND COMPLETED 'REQ-OPER-ROLE';
                          (RESULT: SC2=000, SC1=000, MC=CMD0001); DATE: <date>
  <* %DIAA-000.111017  %  TIA0300 $DIALOG APPLICATION CORRECTLY STARTED ON
                          HOST *STDHOST
  <G %IOR1-000.111019  %  NKR0175 CONFIGURATION UPDATE STARTED.
OPRT /@001-000.111019 SH-DEV A007
@001 +XACM-000.111019 MNEM DEV-TYPE CONF-STATE POOL VSN   DEV-A    PHASE    ACTION
@001 +XACM-000.111019 A007 FTAPE1   DETACHED    NO         FREE
@001 ! UCO-000.111019  %  NBR0740 COMMAND COMPLETED 'SH-DEV';
                          (RESULT: SC2=000, SC1=000, MC=CMD0001); DATE: <date>
OPRT /(CB)-000.111047 ASR ADD,CS=C0,CD=ALL
(CB) ! UCO-000.111047  %  NBR0740 COMMAND COMPLETED 'ASR';
                          (RESULT: SC2=000, SC1=000, MC=CMD0001); DATE: <date>
  <J %OAFW-000.111055  %  JMS0154 'TSOS' LOGGED ON FOR 'PGTD0666/STAT0C33'.
                          JOB NAME 'QE13END'. CALLER '(NONE)'. TID 00020033
OPRT /@001-000.111113 ATT A007
  <G % MSG-000.111113  %  NKR0042 'DEVICE     =A007': ATTACH ACCEPTED
  <G % MSG-000.111113  %  NKR0040 'DEVICE     =A007' ATTACHED
@001 ! UCO-000.111113  %  NBR0740 COMMAND COMPLETED 'ATT';
                          (RESULT: SC2=000, SC1=000, MC=CMD0001); DATE: <date>
  <J %OAFX-000.111122  %  JMS0154 'TSOS' LOGGED ON FOR 'PGTD0666/STAT0C50'.
                          JOB NAME 'QE13END'. CALLER '(NONE)'. TID 00020031
OPRT /@001-000.111128 SH-DEV A007
@001 +XACM-000.111128 MNEM DEV-TYPE CONF-STATE POOL VSN   DEV-A    PHASE    ACTION
@001 +XACM-000.111128 A007 FTAPE1   ATTACHED    NO         FREE
@001 ! UCO-000.111128  %  NBR0740 COMMAND COMPLETED 'SH-DEV';
                          (RESULT: SC2=000, SC1=000, MC=CMD0001); DATE: <date>
  <E %XACO-000.111134  %  NBR0797 APPLICATION '@002' CONNECTED WITH '$CONSOLE',
                          PROCESSOR NAME 'D017ZE14', STATION NAME 'OMS00068'
  <J %OAFZ-000.111134  %  JMS0154 'TSOS' LOGGED ON FOR 'PGTD0666/STAT0C53'.
                          JOB NAME 'QE13END'. CALLER '(NONE)'. TID 0002002C
  <J %OAFY-000.111205  %  JMS0154 'TSOS' LOGGED ON FOR 'MCP0242C/STAT0690'.
                          JOB NAME 'SQ13LUEN'. CALLER '(NONE)'. TID 0002002D
:
```

14.4 RESLOG logging file

BS2000 servers provide the option of exceeding the nominal server performance for a limited time by attaching preinstalled CPUs (“extra CPUs”, see see the “Introduction to System Administration” manual [6]).

The RESLOG subsystem records the attachment and detachment of extra CPUs in the RESLOG log file.

The RESLOG subsystem

The RESLOG (RESource LOGging) subsystem is part of the basic configuration.

In VM2000 mode RESLOG checks if it is running in the monitor system under VM2000 and whether extra CPUs exist.

The RESLOG subsystem offers two command interfaces for working with the RESLOG log file:

Command	Meaning
CHANGE-RESLOG-FILE	Change the log file
START-RESLOG-EVALUATION	Evaluate the log file

The RESLOG log file

RESLOG creates a log file under the TSOS user ID with the following name:

SYS.RESLOG.<server-id>

The <server-id>, derived from the CPU-ID, identifies a server uniquely world-wide. With the SHOW-SYSTEM-INFORMATION command all CPU-IDs of a configuration can be output. They differ from each other only in their serial numbering which is provided at different positions in the CPU ID depending on the architecture. These positions are set to “0”. The result is the server ID.

This log file normally stays open during the entire session. Log data is appended to an existing log file it its server ID is identical.

The log file can be changed with the CHANGE-RESLOG-FILE command.
The current file is then closed and renamed to SYS.RESLOG.<server-id>.<date> with the date when the file was closed. A new file named SYS.RESLOG.<server-id> is then opened.

The log file contains various records:

– When RESLOG is started, the first thing done is to write a start record. The start record contains date, time, OSD version and RESLOG version. It also contains the server ID, the number of extra CPUs and the CPU numbers of the extra CPUs that are ATTACHED.

– A stop record is written as the last record when the subsystem is terminated (meaning at the time of shutdown) and when changing the log file. The stop record contains the time and date as well as the reason for closing the log file (stop/change).

– Every ATTACH-/DETACH-DEVICE or ATTACH-/DETACH-VM-RESOURCES for an extra CPU causes a CPU record with date, time, CPU number and ATTACHED/DETACHED identifier to be written.

– Another record is written or updated once per hour, the alive record, so that the data is still consistent even after a system crash.
  The alive record is normally only located in the open file and is overwritten by a CPU or stop record.
  The only time an alive record can be found as the last record that is not overwritten the next time RESLOG is started is after a system crash.

Evaluating the RESLOG log file

With the START-RESLOG-EVALUATION command you start the RESLOG evaluation. The result can be output in abbreviated or detailed form on the screen or in a file.

The RESLOG evaluation runs as an independent program that is loaded, started and terminated internally.

WARNING!
If the RESLOG evaluation is called from a different program (e.g. after interrupting with the K2 key), then this program is unloaded. You cannot return to this program after terminating the RESLOG evaluation (e.g. with //RESUME).

Systems support defines the amount of data output and the destination of the output from the RESLOG evaluation. The various operands of the START-RESLOG-EVALUATION command are available for this purpose:

– Operand RESLOG-FILE=*CURRENT/<filename>/*FROM-FILE(...)
  The current RESLOG file, a single RESLOG files or a list of RESLOG files can be evalauted.
  The following conditions apply when outputting a list of files:
  – Only RESLOG files from one server (meaning RESLOG files with the same server ID) can be evaluated in an evaluation run.
  – The RESLOG files are to be specified in chronological order, starting with the oldest.
– Operand PERIOD=*INTERVAL(...)
  The evaluation can run the entire time or for a specific period of time.
– Operand INFORMATION=*SUMMARY/*ALL
  The output is a short summary or also contains a list of every single ATTACH/DETACH operation.
– Operand OUTPUT=*SYSOUT/<filename>
  The output is sent to SYSOUT or to a file.

Example 1: Summary output of the current RESLOG log file to SYSOUT

```
/START-RESLOG-EVALUATION [RESLOG-FILE=*CURRENT,INF=*SUMMARY,OUTPUT=*SYSOUT]

------------------------------------------------------------------------------
                            RESLOG EVALUATION
------------------------------------------------------------------------------
START DATE              : <date>      END DATE  : <date>
------------------------------------------------------------------------------
OSD VERSION             : 10.0A00             SERVER-ID : 1002000121900000
RESLOG VERSION          : 01.6A00
------------------------------------------------------------------------------
NUMBER EXTRA CPUS       :       1
NUMBER DAYS IN USE      :       1
------------------------------------------------------------------------------
TIME WITHOUT DATA       :      1 (hours) (2%)
------------------------------------------------------------------------------
```

The following items mean the following:

START DATE	Start of the evaluation period
END DATE	End of the evaluation period
OSD VERSION	Operating system version at the start of the evaluation period
RESLOG VERSION	Version of the RESLOG file at the start of the evaluation period
SERVER-ID	Unique (world-wide) server ID of the server
NUMBER EXTRA CPUS	Number of extra CPUs at the start of the evaluation period
NUMBER DAYS IN USE	Number of days in which the extra CPUs were used; Every day an extra CPU is ATTACHED is counted for every extra CPU.
TIME WITHOUT DATA	Time in hours/days in which RESLOG was not active; The number of hours/days between the STOP and the following START records are counted. The time is also output as a percentage of the total evaluation period.

Example 2:
Detailed output of the current RESLOG log file in the file PROT.EXTRA-CPU.002

```
/START-RESLOG-EVALUATION INFORMATION=*ALL, OUTPUT=PROT.EXTRA-CPU.002
:
```

<output just like for INF=*SUMMARY, see Example 1>:

```
--------------------------------------------------------------------------------
                    DETAILED STATISTICS OF EXTRA-CPUS
--------+-------------------------+-------------------------+-----------------
 CPU NR !      ATTACH TIME        !      DETACH TIME        !     DURATION
--------+-------------------------+-------------------------+-----------------
  02    !  <date>       <time>    !  <date>          <time> !      1 (days)
        !                         !                         !
  02    !  <date>       <time>    !  <date>          <time>*!      1 (days)
        !                         !                         !
NO DATA !  <date>       <time>    !  <date>          <time> !      1 (hours)
--------------------------------------------------------------------------------
```

The following items mean the following:

CPU NR
: CPU number of the extra CPU or – when RESLOG is not loaded – NO DATA

ATTACH TIME/ DETACH TIME
: Time and date of the ATTACH or DETACH procedure for an extra CPU (if there is a CPU number in CPU NR)
or
time and date of the beginning of a period in which RESLOG was not active (if the text NO DATA is found in CPU NR)

DURATION
: Period in hours or days based on the ATTACH/DETACH TIME

  – Output for extra CPU:
    The number of calendar days of the period between the ATTACH and DETACH of the extra CPU is output.
  – Output for NO DATA:
    The number of hours or days (rounded off) of the period in which RESLOG was not active is output.

An asterisk (*) can be output to the right of the times for ATTACH TIME and DETACH TIME with the following meaning:

– The exact time of the ATTACH is not known because the extra CPU was already ATTACHED when RESLOG was started.

– The exact time of the DETACH is not known because the file was closed, the system switched to a new one or the subsystem was terminated abnormally while an extra CPU was ATTACHED.

– The actual time of the ATTACH/DETACH is not output because the evaluation period begins after an ATTACH and/or ends before a DETACH.

– The current file is evaluated and an extra CPU is ATTACHED.
The date of the last alive or CPU record is written under DETACH TIME.

Messages with Remote Service

All ATTACH/DETTACH operations for extra CPUs are sent via the Remote Service to the manufacturer to ensure that the contractual usage period of the extra CPUs can be checked. These messages also serve as a way to check the data recorded by RESLOG. Normally, however, the contractual duration of usage is only checked based on the RESLOG data. You must therefore make sure that RESLOG is always running.

RESLOG also sends messages via Remote Service to the manufacturer when it abnormally terminates due to an error. The following messages are intended to announce this:

```
NPR0001   SUBSYSTEM RESLOG COULD NOT BE INITIALIZED
          Further information see SERSLOG.

NPR0002   SUBSYSTEM RESLOG TERMINATED ABNORMALLY AFTER SUCCESSFUL INITIATION.
          Further information see SERSLOG.
```

Abbreviations

ACS	Alias Catalog Service
AID	Advanced Interactive Debugger
ASE	Auxiliary SERSLOG Extensions
CCB	Channel Control Block
CCW	Channel Command Word
CLTF	Common Log Task Facility
CPU	Central Processing Unit
CSECT	Control SECTion
DAB	Disk Access Buffer
DCAM	Data Communication Access Method
DCM	Data Communication Methods
DMS	Data Management System
DSECT	Dummy SECTion
DSSM	Dynamic SubSystem Management
EDT	BS2000 file editor
ELFE	Error Logging File Evaluation
ELSN	Error Log Sequence Number
EOLDTAB	System module table
ETPND	Identification table of the CP subsystem
EXVT/XVT	eXecutive Vector Table
FCB	File Control Block
FDDRL	Fast Disk Dump and ReLoad
HSA	Hardware System Area
HSI	Hardware-software interface
IPL	Initial Program Loading
ISAM	Indexed Sequential Access Method
ITN	Internal Task Number

JCB	Job Control Block
JTBP	Job-To-Be-Processed block
JTBPX	Job-to-be-processed block Extension
NDM	Nucleus Device Management
PAM	Primary Access Method
PCB	Program Control Block
PSA	Processor Save Area
PSA	Prefixed Storage Area
PSW	Program Status Word
SDITT	Start Device Interrupt Trace Table
SERSLOG	Software Error Logging
SIH	System Interrupt Handling
SIR	System Install and Restore
SLED	Self Loading Emergency Dump
SPL	System Programming Language
SVC	SuperVisor Call
SVMT	System Virtual Memory Table
TCB	Task Control Block
TDL	Trace Dump List
TFT	Task File Table
TIC	Task in Control
TID	Task IDentifier
TLT	Task Location Table
TPR	Task PRivileged
TSN	Task Sequence Number
TTSAV	System trace table
TU	Task Unprivileged
UVMT	User Virtual Memory Table
VAT	Virtual Attribute Table

Related publications

You will find the manuals on the internet at *http://manuals.ts.fujitsu.com*. You can order printed copies of those manuals which are displayed with an order number.

[1] **AID** (BS2000)
Advanced Interactive Debugger
Core Manual

[2] **EDT** (BS2000)
Statements
User Guide

[3] **ELSA** (BS2000)
Error Logging System Analysis
User Guide

[4] **BS2000 OSD/BC**
Executive Macros
User Guide

[5] **BS2000 OSD/BC**
System Installation
User Guide

[6] **BS2000 OSD/BC**
Introduction to System Administration
User Guide

[7] **BS2000 OSD/BC**
Utility Routines
User Guide

[8] **BS2000 OSD/BC**
Commands
User Guide

[9] **SECOS** (BS2000)
Security Control System - Audit
User Guide

Index

R

S